Multivariate Algorithmic Analysis of Hitting Small Sets

Mithilesh Kumar

Dissertation for the degree of philosophiae doctor (PhD)
at the University of Bergen

2017

Scientific environment

The work of this thesis, both research and writing, was done in the algorithms group at the Department of Informatics at the University of Bergen.

Moreover, the project was done entirely under the supervision of Professor Daniel Lokshtanov, and the project was funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 306992 and the Bergen Research Foundation under the Beating Hardness by Preprocessing grant.

Acknowledgments

First and foremost, I would like to thank my supervisor Professor Daniel Lokshtanov for valuable insight and ideas leading to the work presented in this thesis. I gained immensely by his advice, on both academic and non-academic aspects. I'll surely miss his guidance and company in immediate future.

Next I would like to thank Professor Saket Saurabh, the man behind the scene, watching, protecting. Even my position was funded from his ERC grant. I have learnt from him how to do research and this is certainly going to help me in my future endeavors.

I would like to thank Syed Meesum, my friend and brother since my IIT days. It was his suggestion to apply to Bergen for PhD in algorithms. He has remained with me through thick and thin and I'm blessed to have a friend like him. I would like to thank my co-authors Amer Mouawad, Fahad Panolan and Edward Eiben for their support and commitment.

Finally, I'm blessed to have a wonderful family without whose continuous support, this work wouldn't have seen the light of the day.

Abstract

When a problem has been shown to be NP-complete, often one has to be content with either exponential-time algorithms or resort to approximation algorithms that sacrifice the optimality of the solution, or with ad hoc heuristics, which often remain unreliable. In multivariate algorithms, one tries to capture any hidden structure in the input via a set of parameters. For example, for a single parameter encoded by a number k, one tries to find an algorithm whose running time is of the form $f(k) \cdot n^{\mathcal{O}(1)}$. Such an algorithm is called an FPT algorithm. The interesting property of an FPT algorithm is that if k is a fixed constant, or even grows slowly enough with the input size, the algorithm takes polynomial-time asymptotically, regardless of whether the problem is NP-complete or not. In a few decades, it has been shown that a vast number of NP-complete problems admit such algorithms. For example, consider the NP-complete problem called the Vertex Cover problem in which, given a graph on n vertices, the task is to find a set (called vertex cover) of vertices of smallest cardinality whose removal makes the graph edgeless. With the size k of the solution as the parameter, the Vertex Cover problem can be solved in $\mathcal{O}(1.2738^k + k \cdot n)$ time [CKX10]. So, if k is very small compared to n, the Vertex Cover problem can be solved in time linear in the size of the input! Similarly, consider the undirected Feedback Vertex Set problem in which, given an undirected graph, the task is to find a set (called feedback vertex set) of vertices of smallest cardinality whose removal makes the graph acyclic. With the size k of the solution as the parameter, this problem can be solved in $\mathcal{O}(3.592^k n^{\mathcal{O}(1)})$ time [KP14] and the directed version can be solved in time $\mathcal{O}(4^k k^{O(1)} k! nm)$ [CLL$^+$08]. Although different problems come with their own structure and specialized techniques are needed to solve them, there are some generic upper bounding techniques for the running time of FPT algorithms, namely branching, iterative compression and kernelization which we will encounter many times in the thesis.

Furthermore, there are techniques that can be used to classify problems as those that probably do not admit FPT algorithms and to lower bound the dependence of the running time on the parameter. These techniques can only be used to show *conditional* results, because unconditional superpolynomial running time lower bounds for problems in NP would resolve P vs NP problem. The assumption typically used for ruling out FPT algorithms is that FPT $\neq$ W[1], while the assumption used for lower bounding dependencies on k is called the Exponential-time hypothesis(ETH).

In this thesis, we make a contribution towards multivariate analysis of a class of problems known as Hitting Set problems. In a Hitting Set problem, the input consists of a universe U and a family of sets $\mathcal{H}$ and the task is to find a set $H \subseteq U$ such that H contains at least one element of every set in $\mathcal{H}$. Some of the most studied problems that have a natural formulation as a Hitting Set problem are Vertex Cover, Feedback Vertex Set and Dominating Set. In particular, this thesis focuses on the following problems:

- **Feedback Vertex Set for Tournaments**, where a tournament is an orientation of a complete graph.
- **Feedback Vertex Set for Bipartite Tournaments**, where a bipartite tournament is an orientation of a complete bipartite graph.
- **ℓ-Component Order Connectivity**. In this problem, the input is a graph G with integers ℓ and k and the task is to determine whether there exists a set S of vertices of cardinality at most k such that in $G - S$ (graph obtained after deleting vertices from S) the size of components (number of vertices in a connected set) is bounded by ℓ. This problem happens to be a generalization of the Vertex Cover.
- **Connected Dominating Set in sparse graphs**. In the Dominating Set problem, the input is a graph G with an integer k and the task is to determine whether there exists a set S of vertices of cardinality at most k such that every vertex in G has at least one neighbor in S. In the Connected Dominating Set problem, we require that the induced subgraph $G[S]$ is connected. In sparse graphs, we consider the problem on graphs of bounded degeneracy and bounded expansion. These graph classes are defined in Chapter 8.

For the first two problems, we provide a unified novel approach to obtain a fast FPT and exact-exponential-time algorithm with running times $1.6181^k \cdot n^{\mathcal{O}(1)}$ and 1.3821^n, respectively. In the process, we obtain structural results that may be of independent interest. The third problem is a generalization of the Vertex Cover problem and we provide a kernel of size $2\ell k$ which matches the lower bound size for $\ell = 1$, which is the Vertex Cover problem itself. In this work we obtain a weighted Expansion Lemma which may be of independent interest. At the same time, we provide first non-trivial application (after the Namhauser-Trotter Theorem-based kernel for Vertex Cover) of Linear Programming based kernel. Finally, in the fourth problem, we use a new framework for obtaining polynomial kernels called Lossy Kernelization for Connected Dominating Set in graphs of bounded degeneracy and bounded expansion. For both of these problems, under reasonable complexity theoretic assumptions, a polynomial kernel is forbidden.

List of papers

The results of this thesis are based on the following publications:

1. Faster Exact and Parameterized algorithm for Feedback Vertex Set in Tournaments [KL16c].
Mithilesh Kumar and Daniel Lokshtanov. *In 33rd Symposium on Theoretical Aspects of Computer Science STACS 2016, pages 49:1-49:13, 2016 .*

Chapter 5 is based on this result.

2. Faster Exact and Parameterized algorithm for Feedback Vertex Set in Bipartite Tournaments [KL16b].
Mithilesh Kumar and Daniel Lokshtanov. *In* 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS 2016, December 13-15, 2016, Chennai, India, pages 24:1-24:15, 2016 .*

Chapter 6 is based on this result.

3. A 2ℓ-Kernel for ℓ-Component Order Connectivity [KL16a].
Mithilesh Kumar and Daniel Lokshtanov. *In* 11th International Symposium on Parameterized and Exact Computation*, IPEC 2016, August 24-26, 2016, Aarhus, Denmark, pages 20:1–20:14 .*

Chapter 7 is based on this result.

4. Lossy Kernels for Connected Dominating Set on Sparse Graphs.
Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, Fahad Panolan and Sebastian Siebertz. *In 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France, pages 29:1–29:15 .*

Chapter 8 is based on this result.

Contents

Part I

Algorithms for Hitting Set problems: An overview

Chapter 1

Introduction

1.1 Coping with NP-hardness

In an *optimization problem*, one looks for an *optimum* among all feasible solutions. Quite often one can generate the set of all feasible solutions in time roughly proportional to their number and the quality of a feasible solution can be computed in polynomial time. For such problems, one approach is to do an exhaustive search over all feasible solutions. Frequently, such search spaces are exponential in the size of the input, which implies an exponential-time for the exhaustive search. In particular, for problems classified as NP-hard, it is believed that there are no algorithms that handle all instances substantially faster than performing an exhaustive search. We'll describe various methods to cope with this issue of exhaustive search. To do this, we will use Vertex Cover as a running example, one of the most fundamental NP-hard problem. We assume basic familiarity with graph theory. We have tried to follow the notation consistent with Diestel's book on Graph Theory [Die12].

The input to the Vertex Cover problem is an undirected graph $G = (V, E)$ with n vertices and the task is to find a smallest cardinality subset S (called a vertex cover) of vertices V such that removing S from the graph G makes it edgeless (i.e. $G \setminus S$ is an independent set). The search space for this problem is 2^V, i.e. the set of all subsets of V. But, when the input graph has some structure, one can use it to guide the search. For example, when the input graph is a tree, one can find a vertex cover of minimum size in $\mathcal{O}(n)$ time: repeatedly remove the set of vertices which are leaves and pick their parents into the solution until the graph becomes empty. Unfortunately, it is unlikely that there exists an algorithm that runs in polynomial time on general graphs (for all possible input graphs). Most approaches to cope with this hurdle can be grouped into two categories: Given an instance of an NP-complete problem, (a) how fast can we solve this instance optimally? and (b) what can any algorithm *do* (making the problem simpler via some preprocessing or obtaining a solution closer to an optimal solution) in polynomial time? Let us briefly outline some algorithms to illustrate the above approaches for the Vertex Cover problem.

Exact exponential-time algorithms. It is always possible to begin with an algorithm $\mathcal{A}_1$ for Vertex Cover that enumerates all subsets of vertices according to their cardinality in nondecreasing order and returns the first subset whose removal makes the graph edgeless. This algorithm would take $2^n \cdot n^{\mathcal{O}(1)}$ time. The class of algorithms that solve the problem optimally and whose running times are expressed as an exponential function of the input size only are known as exact-exponential-time algorithms. The algorithm $\mathcal{A}_1$ is one such algorithm. By relaxing the requirement of optimality or via additional structural information about the input, one can often get algorithms with improved performance.

Parameterized algorithms. Suppose we are not satisfied with solutions of size larger than (say) k, so if the size of the smallest one is more than k, we don't really care what it is, knowing that it is more than k is enough. In this setting, we can get a better algorithm by only trying subsets of size at most k. This reduces the running time to $\mathcal{O}(\binom{n}{k}) = \mathcal{O}(n^k)$ which is polynomial for fixed values of k. But as k gets larger, the running time gets worse. We can do better by considering another algorithm $\mathcal{A}_2$ that takes in addition to G, an integer k as input and outputs a solution only if there exists a solution of size at most k. We start with observing that if G does not contain any vertex of degree at least 2, then we can find an optimal solution in polynomial time. If there is a vertex v of degree at least 2, then we can divide our problem into two independent subproblems. In the first subproblem, the input graph is obtained by assuming that v is in the sought solution H and deleting v from the graph G and decreasing k by 1. In the second subproblem, the input graph is obtained by assuming that v is not in the sought solution H and deleting $N[v]$ from G. In this case, we include $N(v)$ into the solution H and decreasing k by $|N(v)|$. In the first subproblem, the size of the parameter decreases by 1 and in the second subproblem, the size of the parameter decreases by at least 2 implying that $\mathcal{A}_2$ can take at most $1.618^k \cdot n^{O(1)}$ time. The algorithm $\mathcal{A}_2$ belongs to the class of *fixed-parameter-tractable* (FPT) algorithms. Informally, in exact-exponential-time algorithms, the running time is measured solely in terms of the input size n. In FPT algorithms, the running time is expressed as $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f that depends only on the parameter k.

Approximation algorithms. Both algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ take exponential-time, and are guaranteed to output an optimal solution to every input instance. As noted earlier, if we restrict the time to be polynomial in n and k, we don't hope to have an algorithm for Vertex Cover that outputs an optimal solution. Still it is desirable to get a feasible solution as close to optimal as possible. Our next algorithm $\mathcal{A}_3$ outputs a feasible vertex cover, although not necessarily optimal. It finds an inclusion maximal matching M in G in polynomial time and returns the vertex set S incident to the edges of M as a solution. Clearly, S is a vertex cover: $G - S$ is edgeless. As any optimal solution must contain at least one endpoint of each edge in M, the cardinality of S is at most 2 times the cardinality

of any optimal solution. An algorithm of type $\mathcal{A}_3$ is known as an approximation algorithm.

Kernelization. It is natural to wonder what makes Vertex Cover a hard problem. Phrased differently – what do computationally hard instances of Vertex Cover look like? If an instance consists of a *hard* and an *easy* part, is it possible to efficiently identify, solve and remove the easy part, and then leave the hard part for the brute force search? Kernelization is the paradigm of identifying and solving *this easy* part. Informally, in the context of Vertex Cover, kernelization (or kernel) is a polynomial time algorithm, that given an instance (G, k) of Vertex Cover outputs an equivalent instance (G', k'). Here by equivalent instance, we mean that G has a vertex cover of size at most k if and only if G' has a vertex cover of size at most k'. Moreover, the size of the graph G' (in terms of number of vertices or edges) and k' is bounded by $f(k)$ for some computable function f. Let us consider a kernelization algorithm $\mathcal{A}_4$ for the Vertex Cover problem. We start with $G' = G$ and $k' = k$. The main idea in $\mathcal{A}_4$ is to decide in polynomial time whether any vertex v must go into every optimal solution or not. To that end, $\mathcal{A}_4$ applies the following two rules exhaustively: (1) If G' has an isolated vertex v, then remove v from G'. (2) If G' has a vertex v of degree at least $k' + 1$ and $k' \geq 0$, then remove v from G' and decrease k' by 1. If $k' < 0$, then conclude that G does not have any vertex cover of size at most k. As each application of any of the above rules decreases the size of G' by 1, $\mathcal{A}_4$ must terminate in polynomial time. It is easy to see that the rule (1) corresponds to the conclusion that v does not belong to any optimal solution and the rule (2) corresponds to the conclusion that v must belong to every optimal solution. Let us now analyze the size of G': If G' were to have a vertex cover S of size at most k' and by the above rules, the degree of any vertex in G' is at most k', the number of vertices can be at most the total number of vertices in S and in the neighborhood of S, i.e. $k' + k' \times k' = k'(k' + 1) \leq k(k + 1)$. Hence, in polynomial-time using $\mathcal{A}_4$, it is possible to decrease the number of vertices to $\mathcal{O}(k^2)$ which can be significantly smaller than n.

We have seen four of the most commonly used paradigms in theoretical computer science to cope with NP-hardness. In this thesis, we focus on parameterized algorithms and kernelization for a few fundamental problems and give algorithms that significantly improve previously known algorithms. As a common starting point, it is observed that the problems considered in this thesis can be formulated as a special cases of the Hitting Set problem. In the Hitting Set problem, the input is a set of elements, called universe U, and a family $\mathcal{H}$ of subsets of U, and the task is to check whether there exists a subset $H \subseteq U$ of size at most k such that every set in $\mathcal{H}$ has non-empty intersection with H. Hitting Set is a famous NP-complete problem and has been extensively studied from the perspective of parameterized algorithms, exact algorithms, kernelization and approximation algorithms (See Chapter 3 and Chapter 4). The known algorithms for this problem automatically provide an algorithm for the problems considered in this thesis. For example, the Vertex Cover problem can be seen as a Hitting

Set problem in which the universe is the vertex set of the graph and the edge set of the graph is the family of sets. A vertex cover of the graph is a subset of vertices that *hits* every edge. So we can formulate the Vertex Cover problem as this Hitting Set problem and this automatically gives an algorithm for the Vertex Cover problem. But, usually this generic approach does not yield the best possible algorithm for specific problems. Better approaches are designed to utilize more structure from the specific problem and its instances.

1.2 Organization of the thesis

In the next chapter, we provide some essential preliminaries and notations. In Chapters 3 and 4 we provide a survey of results known for Hitting Set problems. These results provide off-the-shelf algorithms for problems that can be formulated as a special case of the Hitting Set problem. A special variant of Hitting Set is the d-Hitting Set problem in which the sets in the family $\mathcal{H}$ are of cardinality at most d. For example, consider graphs known as tournaments which are orientations of complete graphs. In the Feedback Vertex Set problem, the task is to determine whether there exists a set of vertices of size at most k such that its removal makes the graph acyclic. In the case of tournaments, one can formulate this problem as 3-Hitting Set and hence can immediately show that the problem admits an $\mathcal{O}(k^3)$-size vertex kernel and it is fixed-parameter tractable with a simple $3^k \cdot n^{\mathcal{O}(1)}$ time algorithm. Although it's nice to know that the problem is fixed-parameter tractable and has a polynomial kernel, these algorithms are not the best one can achieve for the Feedback Vertex Set problem in tournaments. In chapter 5, we obtain the fastest known parameterized and exact exponential-time algorithm for this problem. Along the way, we obtain a structural result which may be of independent interest. Similarly, when we change the input to a bipartite tournament (a complete directed bipartite graph), the problem can be formulated as a special case of the 4-Hitting Set problem and hence we would get an $\mathcal{O}(k^4)$-size vertex kernel and a simple $4^k \cdot n^{\mathcal{O}(1)}$ time algorithm. In chapter 6, we extend the techniques used in chapter 5 to obtain the fastest known parameterized and exact exponential-time algorithm when the input graph is a bipartite tournament. Another one of the most widely studied problems is the Vertex Cover problem which we have discussed in the introduction. In chapter 7, we study a generalization of this problem known as ℓ-Component Order Connectivity problem in which the input is a graph G and integers k, ℓ and the task is to determine whether there exists a set S of vertices of size at most k such that in $G - S$, the size of the connected components is bounded by ℓ. When $\ell = 1$, each component would be of size 1 which implies that 1-Component Order Connectivity is equivalent to Vertex Cover problem. The ℓ-Component Order Connectivity can be seen as a special case of the $(\ell + 1)$-Hitting Set problem, which immediately implies a kernel of size $\mathcal{O}(k^{\ell+1})$. In chapter 7, we obtain a kernel of size at most $2\ell k$. At the same time, we obtain a generalization of the Expansion Lemma which is widely used in the field of kernelization. The

field of kernelization does not combine very well with the field of approximation algorithms in that an approximation algorithm for a *kernelized* instance does not necessarily provide the same approximation guarantee in the original instance. Another issue that the field of kernelization has faced in recent years is that many important problems like Dominating Set do not admit polynomial kernel in general. We discuss a framework of Lossy kernelization introduced by Lokshtanov et al [LPRS16], which tries to address these issues. Connected Dominating Set can been seen as an implicit hitting set in which the universe is the set of vertices and the family is the set of closed neighbourhoods of vertices. The required hitting set for the family must induce a connected set in the graph. We provide lossy kernels for Connected Dominating Set in graphs of bounded degeneracy and bounded expansion. In chapter 9, we provide some further direction for future research.

Chapter 2

Preliminaries and notations

2.1 Graph terminalogy

Let $\mathbb{N}$ denote the set of positive integers $\{0, 1, 2, \dots\}$. For any non-zero $t \in \mathbb{N}$, $[t] := \{1, 2, \dots, t\}$. We denote a constant function $f : X \to \mathbb{N}$ such that for all $x \in X, f(x) = c$, by $f = c$. For any function $f : X \to \mathbb{N}$ and a constant $c \in \mathbb{N}$, we define the function $f + c : X \to \mathbb{N}$ such that for all $x \in X, (f + c)(x) = f(x) + c$. We use the same symbol f to denote the restriction of f over a subset of it's domain, X. For a set $\{v\}$ containing a single element, we simply write v.

In this thesis, we have tried to follow the standard graph theoretic notation consistent with Diestel's book on Graph Theory [Die12]. A graph is defined as $G := (V, E)$. We use $V(G)$ and $E(G)$ to denote the set of vertices and edges of G, respectively. A vertex $u \in V(G)$ is said to be **incident** on an edge $e \in E(G)$ if u is one of the endpoints of e. A pair of edges $e, e' \in E(G)$ are said to be **adjacent** if there is a vertex $u \in V(G)$ such that u is incident on both e and e'. For any vertex $u \in V(G)$, by $N(u)$ we denote the set of **neighbors** of u i.e. $N(u) := \{v \in V(G) \mid uv \in E(G)\}$. For any subgraph $X \subseteq G$, by $N(X)$ we denote the set of neighbors of vertices in X outside X, i.e. $N(X) := (\bigcup_{u \in X} N(u)) \setminus X$. A pair of vertices $u, v \in V(G)$ are called **false twins** if $N(u) = N(v)$. An induced subgraph on $X \subseteq V(G)$ is denoted by $G[X]$. For a graph G and a vertex v, $G - v$ denotes the graph obtained from G by deleting v and all edges incident to v. Similarly, for a vertex set S, $G - S$ denotes the graph obtained from G by deleting all vertices in S and all edges incident to them.

A **path** P is a graph, denoted by a sequence of vertices $v_1 v_2 \dots v_t$ such that for any $i, j \in [t], v_i v_j \in E(P)$ if and only if $|i - j| = 1$. A **cycle** C is a graph, denoted either by a sequence of vertices $v_1 v_2 \dots v_t$ or by a sequence of edges $e_1 e_2 \dots e_t$, such that for any $i, j \in [t]$ $u_i u_j \in E(C)$ if and only if $|i - j| = 1 \mod t$ or in terms of edges, for any $i, j \in [t]$, e_i is adjacent to e_j if and only if $|i - j| = 1 \mod t$. Simply put, a cycle as a path whose endpoints are connected by an edge. The **length** of a path (cycle) is the number of edges in the path(cycle). A **triangle** is a cycle of length 3. In G, for any pair of vertices $u, v \in V(G)$ $\mathsf{dist}(u, v)$ represents the length of a shortest path between u and v. A **tree** is a connected graph that does not contain any cycle. A **rooted tree** T is a tree with a special vertex r

called the root of T. With respect to r, for any edge $uv \in E(T)$ we say that v is a child of u (equivalently u is parent of v) if $\mathsf{dist}(u,r) < \mathsf{dist}(v,r)$. A **forest** is a collection of trees. A **rooted forest** is a collection of rooted trees. A **clique** is a graph that contains an edge between every pair of vertices.

In a directed graph D, the set of **out-neighbors** of a vertex v is defined as $N^+(v) := \{u \mid vu \in E(D)\}$. Similarly, the set of **in-neighbors** of a vertex v is defined as $N^-(v) := \{u \mid uv \in E(D)\}$. For any set of edges C (directed or undirected) and set of vertices X, the set $V_C(X)$ represents the subset of vertices of X which are incident on an edge in C. For a vertex $v \in V(G)$, the set $N_C(v)$ represents the set of vertices $w \in V(G)$ such that there is an undirected edge $wv \in C$.

2.2 General Background

In the thesis, we provide a brief survey of results for problems related to Hitting Set. We mention results from the perspective of exact-exponential-time algorithms, approximation algorithms, parameterized algorithms and kernelization. In the introduction, we used Vertex Cover to illustrate these four methodologies. For more detail on exact-exponential-time algorithms, we refer the reader the book of Fomin et al [FK10]. In this thesis, we assume basic familiarity with complexity theory, in particular with classes P, NP, coNP and NP-hard. The reader is advised to refer to the excellent book on Complexity theory by Arora and Barak [AB09]. For approximation algorithms, the reader is referred to books like [WS11, Vaz03]. For parameterized complexity and kernelization the reader is referred to books like [FG06, DF13, CFK$^+$15]. In this section, we briefly describe the Unique Games Conjecture which we mention often throughout the thesis.

Unique games conjecture

Researchers have failed to find algorithms for certain problems (in particular belonging to the class NP-hard) that given any instance of these problems output a solution within a specified factor. For example, we do not know of any algorithm for Vertex Cover that given an instance of Vertex Cover outputs a vertex cover within a factor of 2 and runs in polynomial time. Dinur and Safra [ID05] proved that Vertex Cover can not be approximated within a factor of 1.3606 unless $\mathsf{P} = \mathsf{NP}$. Unfortunately, inapproximability proved using their method is not tight. Subhash Khot in 2002 [Kho02] conjectured the inapproximability of a problem known as the Unique Label Cover problem. This conjecture is known as the Unique games conjecture (UGC). Using this assumption, researchers have been able to prove much tighter inapproximability results. We briefly describe this conjecture. First, we define the following problem.

LABEL COVER

Input:	A graph G, a set of labels L and for each edge $e = (u, v) \in E$, a set $R_e \subseteq L \times L$ consisting of a set of "permissible" values for the pair (u, v).
Question:	Does there exists an assignment of labels l_u to each vertex u in G such that at least α fraction of edges are satisfied where an edge $e = (u, v)$ is satisfied if $(l_u, l_v) \in R_e$?

Let $Val(LC) \in [0, 1]$ denote the maximum possible fraction of satisfied edges by any labeling. Let us consider an example in which we formulate 3-COLORING as a LABEL COVER problem. In the 3-COLORING problem, input is a graph G and the task is to check whether vertices of G can be colored using at most 3 colors such that for none of the edges in G, its endpoints are assigned the same color. As a LABEL COVER problem, we work with the same graph G. The labels L are the three colors, i.e. $L = \{1, 2, 3\}$. For each edge e, the set of color pairs is the same, i.e. $R_e = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$. It is easy to see that $Val(LC) = 1$ if and only if G is 3-colorable. Since, 3-COLORING problem is NP-complete, LABEL COVER is NP-complete as well. Using classical complexity theory, it was known that

Theorem 1. *For every $\epsilon > 0$ there exists an L such that it is NP-hard to distinguish* LABEL COVER *instances, LC, with $Val(LC) = 1$ from those with $Val(LC) \leq \epsilon$ for instances LC with the provided label set L.*

In the UNIQUE LABEL COVER problem, the relation R_e for each edge is a bijection i.e. for each edge $e = (u, v)$ and choice of label for u there is exactly one choice of label for v that satisfies the edge e.

Conjecture 2.1 (Unique Games Conjecture [Kho02])**.** *For any $\epsilon > 0$ there exists an L such that it is NP-hard to distinguish* UNIQUE LABEL COVER *instances with $Val(ULC) > 1 - \epsilon$ from those with $Val(ULC) \leq \epsilon$ for instances ULC with the provided label set L.*

Despite continuous efforts to prove or disprove UGC, there is still no consensus regarding its validity. UGC implies that the currently best known approximation algorithms for many important computational problems have optimal approximation ratios.

2.3 Multivariate Analysis

Multivariate analysis began as a field in 1980s with the work of Downey and Fellows, although some isolated results in the context of integer linear programming and graph minor theory where already known. Now, multivariate analysis has matured, is taught as courses in universities and there are multiple books available [FG06, DF13, CFK+15]. Next, we provide formal definitions and state

crucial results in this field. Most of the text is taken from the book Parameterized algorithms [CFK$^+$15][marked with $\star$].

Fixed-Parameter Tractability

To define a parameterized problem, we require to augment the description of a classical problem by an integer.

Definition 2.2 ($\star$)**.** A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*.

For example, x can be an encoding of a graph G and an integer k may monitor some property of the graph or the sought solution like G has vertex cover of size at most k or the size of the solution is at most k.

Definition 2.3 ($\star$)**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a *fixed-parameter algorithm*), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

Given to computational tasks A and B, a reduction from A and B (on vice versa) maintains the equivalence of solutions. In the parameterized setting, one needs to keep track of parameters that come along A and B. Similar to polynomial-time reductions in classical complexity theory, parameterized reductions are defined.

Definition 2.4 (Parameterized Reduction $\star$)**.** Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', y') of B such that

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

The class FPT is closed under parameterized reductions, i.e.

Theorem 2 ($\star$)**.** *If there is a parameterized reduction from A to B and B is FPT, then A is FPT as well.*

Theorem 3 ($\star$)**.** *If there are parameterized reductions from A to B and from B to C, then there is a parameterized reduction from A to C.*

W-hardness

Not every problem is FPT. To characterize problems that do not seem to have algorithms whose running times are of the form $f(k)\cdot n^{\mathcal{O}(1)}$, Downey and Fellows [DF99] defined what is called W-hierarchy.

$$\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \cdots$$

Here W[1] is the analogue of NP, in the sense that problems hard for W[1] are considered unlikely to be in FPT. Informally, W[1] is a class of problems that reduce to Independent Set problem in a parameter-preserving way. In the Independent Set problem, the input is a graph G and an integer k and the task is to determine whether there exists a set S of vertices of size at least k such the graph induced on S has no edge. The above hierarchy is conjectured to be strict i.e. we do not expect an algorithm whose running time is of the form $f(k) \cdot n^{\mathcal{O}(1)}$ for any problem that is hard for any class W[t] where $t \geq 1$.

Independent Set

Input:	A graph G and an integer k
Question:	Does G has an independent set of size at least k?

Dominating Set

Input:	A graph G and an integer k
Question:	Does G has a subset $S \subseteq V(G)$ of size at most k such that every vertex in $V(G) \setminus S$ has at least one neighbor in S?

Theorem 4 ($\star$)**.** Independent Set *and* Clique *are* W[1]*-complete, while* Dominating Set *is* W[2]*-complete.*

Using Theorem 4, we conclude that Independent Set and Dominating Set does not have any algorithm that runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ for any function f that depends on k only. To prove tighter lower bounds, the Exponential time hypothesis [IP99] and the Strong Exponential time hypothesis [IPZ01] are used.

CNF-SAT

Input:	A propositional formula φ on n Boolean variables $x_1, \ldots, x_n$ in conjunctive normal form
Question:	Does there exist an assignment of true/false values to the variables so that φ becomes true?

In the q-SAT problem is CNF-SAT in which the number of variables in each clause is bounded by some constant q. For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists an algorithm solving q-SAT in time $\mathcal{O}^*(2^{cn})$.

Conjecture 2.5 (Exponential-Time Hypothesis, ETH $\star$)**.**

$$\delta_3 > 0$$

Conjecture 2.6 (Strong Exponential-Time Hypothesis, SETH $\star$)**.**

$$\lim_{q\to\infty} \delta_q = 1$$

One can give a reduction from CNF-SAT to Independent Set showing that Independent Set can not be solved in $2^{o(n)}$ time and in $f(k) \cdot n^{o(k)}$ time unless ETH fails.

Kernelization

We already saw an example of kernelization algorithm for Vertex Cover. In this section, we formally define what is a kernel or a kernelization algorithm. We say that two instances of a parameterized problem Q are *equivalent* if $(I, k) \in Q$ if and only if $(I', k') \in Q$. Our task in kernelization is to given an instance $(I, k) \in Q$, apply what is called *data reduction rules* to reduce it to an equivalent instance $(I', k') \in Q$ such that it becomes *easier* to solve (I', k').

Definition 2.7 ($\star$)**.** A *data reduction rule* for a parameterized problem Q is a function $\varphi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that φ is computable in time polynomial in $|I|$ and k.

Definition 2.8 (Safeness $\star$)**.** A data reduction rule φ is called *safe* if it translates an instance to an equivalent one.

Definition 2.9 (Size $\star$)**.** The *output size* of a preprocessing algorithm $\mathcal{A}$ is a function $size_{\mathcal{A}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined as follows:

$$size_{\mathcal{A}}(k) = sup\{|I'| + k' : (I', k') = \mathcal{A}(I, k), I \in \Sigma^*\}.$$

Definition 2.10 (Kernelization, kernel $\star$)**.** A *kernelization algorithm* or simply a *kernel*, for a parameterized problem Q is an algorithm $\mathcal{A}$ that, given an instance (I, k) of Q, works in polynomial time and returns an equivalent instance (I', k') of Q. Moreover, we require that $size_{\mathcal{A}} \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

By above definition, if a problem Q admits a kernel, then the output instance (I', k') has size $g(k)$. Clearly a brute-force-search for solution to (I', k') would run in time $f(g(k))$ implying that Q is in FPT. The following theorem formalizes this intuition.

Theorem 5. *A decidable parameterized problem Q is* FPT *if and only if it admits a kernelization algorithm.*

Therefore, for parameterized problems, it is typical to ask whether the problem is FPTand if yes, ask whether there is polynomial kernel. If one is not able to find any polynomial kernel, it could be because there does not exist any polynomial kernel. There is a framework to prove non-existence of such kernels (see Chapter 15 in [CFK$^+$15]).

Chapter 3

Hitting Set

In the Hitting Set problem, we are given a set U (called *Universe*) and a family $\mathcal{H}$ of subsets of U, and are asked to find a set $H \subseteq U$ of size at most k such that every set in $\mathcal{H}$ has at least one element in H. The set H is said to *hit* every set in the family $\mathcal{H}$. A plethora of problems can be formulated as a Hitting Set problem. One of the most widely studied problems in graph algorithms is the Dominating Set problem, in which the input is a graph G with an integer k and the task is to determine whether there is a subset H of vertices of size at most k such that for every vertex v in the graph, either v is contained in H or H contains a neighbor of v. In this case, the universe is $V(G)$ and the family $\mathcal{H}$ is the set of closed neighborhoods of every vertex in $V(G)$. Then, a hitting set H of $\mathcal{H}$ has the property that for every vertex v, either v belongs to the hitting set H or H contains a neighbor of v implying that H is a dominating set in G. Another widely studied problem is the Feedback Vertex (Edge/Arc) Set problem, in which the input is a (directed or undirected) graph G with an integer k and the task is to determine whether there is a subset H of vertices (or edges) of size at most k such that $G - H$ is an acyclic graph. In this case, the universe is the set of vertices (or edges) and the family $\mathcal{H}$ is the set of all cycles in the graph given by its set of vertices (or edges). Note that in this context, the family $\mathcal{H}$ is given *implicitly*. We will discuss about implicit hitting sets in Chapter 4. A feedback vertex (arc) set is a set of vertices (or edges) that *hits* every cycle in the graph. Another example is the Minimum Spanning Tree problem. Given a connected graph $G = (V, E)$ with positive edge weights $w_e, e \in E$, find a spanning tree with minimum sum of edge weights. Given a graph $G = (V, E)$, a cut is defined by a subset $S \subseteq V$ of vertices, and consists of all the edges that connect elements of S with elements of $V \setminus S$. In this case, $U = E$ and $\mathcal{H}$ is the family of *cuts* in the graph. The task is to compute a subset of E of minimum weight that hits every cut in $\mathcal{H}$. If a set of edges hits all cuts, it defines a connected, spanning subgraph. Minimizing the sum of edge weights ensures that we actually get a tree -the minimum spanning tree. In this chapter, our goal is to survey some of the algorithms for the Hitting Set problem. Later, we'll show how this serves as a starting point for many problems that can be formulated as a Hitting Set problem.

3.1 General Hitting Set

We begin by defining the *decision* version of the HITTING SET problem that takes an additional input k.

HITTING SET

Input:	A family of sets $\mathcal{H}$ of a universe U and an integer k
Question:	Is there a set $H \subseteq U$ of size at most k such that the intersection of H with any set in $\mathcal{H}$ is non-empty?

A set H that intersects all sets in $\mathcal{H}$ is called a *hitting set* of $\mathcal{H}$. We now discuss a problem known as SET COVER which is closely related to HITTING SET.

Relation to Set Cover. In the SET COVER problem, the input is the same as that for the HITTING SET problem, but the task changes to determining whether there exists a subfamily $\mathcal{C} \subseteq \mathcal{H}$ of size at most k such that the union of sets in $\mathcal{C}$ is equal to the universe U, i.e. every element in U is *covered* by $\mathcal{C}$. Since both the problems have the same input, one can construct a bipartite set-element incidence graph $G = (A, B, E)$ such that A contains a vertex for every element in the universe U and B contains a vertex for every set in the family $\mathcal{H}$. For any pair of vertices $a \in A, b \in B$, there is an edge $ab \in E$ if and only if the element corresponding to a is contained in the set corresponding to b. A solution for HITTING SET corresponds to a subset $S_A \subseteq A$ such that every vertex in B has at least one neighbor in S_A. Whereas a solution for the SET COVER problem corresponds to a subset $S_B \subseteq B$ such that every vertex in A has at least one neighbor in S_B. After a moment's reflection, one can verify that using the bipartite set-element incidence graph, one can reduce the HITTING SET problem to the SET COVER problem (and vice versa) with a family of sets $\mathcal{U}$ containing a set for each element in U and the universe H containing an element for each set in $\mathcal{H}$. Hence, any algorithm for say, SET COVER can be seen as an algorithm for HITTING SET with roles of $|U|$ and $|\mathcal{F}|$ interchanged. For example, inapproximability of SET COVER with better than a $\mathcal{O}(\log |U|)$-factor implies inapproximability of HITTING SET with better than a $\mathcal{O}(\log |\mathcal{F}|)$-factor.

NP-completeness of Hitting Set Let $n = |U|$ and $m = |\mathcal{H}|$. An optimal hitting set H can be obtained by iterating over all subsets X of U and checking whether X hits every set in $\mathcal{H}$. Such an algorithm will take $\mathcal{O}(2^n \cdot nm)$ time. But this algorithm is exponential in n and the natural question is can one get a polynomial-time algorithm for the HITTING SET problem. Unfortunately, HITTING SET belongs to Karp's list of 21 NP-complete problems [Kar72]. Hence, unless P=NP, there does not exist an algorithm that solves every HITTING SET instance in polynomial time. Next, we briefly sketch an NP-completeness proof for the HITTING SET problem. We reduce from the CNFSAT problem. In a HITTING SET instance, given a subset $H \subseteq U$ of the universe, it is immediate that one can verify whether H is a hitting set in polynomial time. Hence, HITTING SET belongs

to the class NP. Note that the cardinality of the family $\mathcal{H}$ can be exponential in the size of the universe U. The above verification is polynomial in $|U|$ and $|\mathcal{F}|$. Given a formula φ with m clauses on n variables, we construct an instance of Hitting Set. The universe U is a set of size $2n$ containing one element for each literal x and one element for its negation $\neg x$. The family $\mathcal{H}$ is obtained by making one set for each clause that contains elements corresponding to literals in the clause. In addition, to force that for each variable x at least one element corresponding to x or $\neg x$ is picked in the hitting set, we include a set $\{x, \neg x\}$ for every variable x and set $k = n$ which concludes the construction of the Hitting Set instance. A hitting set of the constructed family yields an assignment to the formula such that every literal corresponding to elements in the hitting set is assigned value 1, otherwise assigned value 0.

Exact-exponential-time algorithms and lower bounds. Given that a polynomial-time algorithm is unlikely, can we improve the exponential factor 2^n in the running time? Equivalently, are there infinitely many Hitting Set instances that essentially require search over all subsets of U to find an optimal solution? Informally, the Strong Exponential Time Hypothesis (SETH) states that there does not exist any algorithm that solves every instance of CNF-SAT on n variables in $\mathcal{O}^*((2-\epsilon)^n)$ (where $*$ notation hides polynomial factors) for any $\epsilon > 0$ for every n. Assuming SETH, there is no algorithm for the Hitting Set problem that achieves running time $\mathcal{O}^*((2-\epsilon)^n)$ for any $\epsilon > 0$, where n is the size of the universe [CDL+16]. If one considers the family $\mathcal{H}$ as the parameter, there is an algorithm for Set Cover whose running time has a factor 2^m. Using dynamic programming, the Set Cover problem can be solved in $\mathcal{O}(2^n \cdot nm)$ time where $n = |U|$ and $m = |\mathcal{H}|$ (Fomin et al [FKW04]). Next, by rephrasing the algorithm of Fomin et al, we obtain an algorithm for the Hitting Set that runs in $\mathcal{O}(2^m \cdot nm)$ time.

Theorem 6. *Given a* Hitting Set *instance* $(U, \mathcal{H})$*, the minimum possible size of a hitting set* $S \subseteq U$ *of* $\mathcal{H}$ *can be found in time* $2^{|\mathcal{H}|}(|U| + |\mathcal{H}|)^{\mathcal{O}(1)}$.

Proof. Let $n = |U|$ and $m = |\mathcal{H}|$. Consider an enumeration of $U = \{v_1, v_2, \ldots, v_n\}$. The rows of the dynamic-programming table are indexed by the subsets $\mathcal{X} \subseteq \mathcal{H}$ and the columns are indexed by the elements in U. The table entry for $(\mathcal{X}, j)$ represented by $T[\mathcal{X}, j]$, is defined as the minimum size of a hitting set $X \subseteq \{v_1, v_2, \ldots, v_j\}$ of $\mathcal{X}$. In the base case, $T[\emptyset, 0] = 0$ and $T[\mathcal{X}, j] = +\infty$ for every non-empty subset $\mathcal{X} \subseteq \mathcal{H}$. The recurrence relation for evaluating $T[\mathcal{X}, j]$ is obtained by considering whether v_j is used to hit any set in $\mathcal{X}$.

$$T[\mathcal{X}, j] = \min(T[\mathcal{X}, j-1], 1 + T[\mathcal{X}', j-1]) \tag{3.1}$$

where $\mathcal{X}' = \mathcal{X} \setminus \{F \mid v_j \in F \in \mathcal{H}\}$. After the table has been filled, the algorithm outputs the value $T[\mathcal{H}, n]$. Since the size of the table is $2^{|\mathcal{H}|+1} \cdot |U|$ the above recurrence can be solved in $(|U| + |\mathcal{H}|)^{\mathcal{O}(1)}$ time, we obtain the required running time.

We now prove the correctness of the recurrence relation 3.1, from which the correctness of the algorithm follows. It is easy to see that if $T[\mathcal{X}, j]$ is ∞ i.e. $\mathcal{X}$

can not be hit using vertices in $\{v_1, v_2, \ldots, v_j\}$, $T[\mathcal{X}, j-1]$ and $T[\mathcal{X}', j-1]$ are ∞ as well. Let $X \subseteq \{v_1, v_2, \ldots, v_j\}$ be an optimal hitting set for $\mathcal{X}$. If $v_j \notin X$, then $X \subseteq \{v_1, v_2, \ldots, v_{j-1}\}$ and hits $\mathcal{X}$. Hence, X is a candidate to be considered when computing $T[\mathcal{X}, j-1]$ i.e. $T[\mathcal{X}, j-1] \leq T[\mathcal{X}, j]$. In case $v_j \in X$, then $X \setminus v_j$ hits $\mathcal{X}' = \mathcal{X} \setminus \{F \mid v_j \in F\}$, which implies $T[\mathcal{X}', j-1] \leq T[\mathcal{X}, j] - 1$. Hence, $\min(T[\mathcal{X}, j-1], 1 + T[\mathcal{X}', j-1]) \leq T[\mathcal{X}, j]$.

For the other direction, if both $T[\mathcal{X}, j-1]$ and $T[\mathcal{X}', j-1]$ are ∞, then $T[\mathcal{X}, j]$ will be ∞ as well. If $T[\mathcal{X}, j-1]$ is infinite and $T[\mathcal{X}', j-1]$ is finite, then v_j must be picked to hit $\mathcal{X}$. Let S' be an optimal hitting set of $\mathcal{X}'$ considered in the computation of $T[\mathcal{X}', j-1]$. Then, $S' \cup \{v_j\}$ is a hitting set of $\mathcal{X}$ and is a subset of $\{v_1, v_2, \ldots, v_j\}$. Hence, $S' \cup \{v_j\}$ must be considered in the computation of $T[\mathcal{X}, j]$ which implies that $T[\mathcal{X}, j] \leq \min(T[\mathcal{X}, j-1], 1 + T[\mathcal{X}', j-1])$.

Now we assume that $T[\mathcal{X}, j-1]$ is finite which implies that $T[\mathcal{X}', j-1]$ is finite as well. Let S' be an optimal hitting set of $\mathcal{X}'$ considered in the computation of $T[\mathcal{X}', j-1]$ and let S be an optimal hitting set of $\mathcal{X}$ considered in the computation of $T[\mathcal{X}, j-1]$. Clearly, both S and $S' \cup \{v_j\}$ will be considered for the computation of $T[\mathcal{X}, j]$ as $T[\mathcal{X}, j]$ is the minimum size over all sets which are subsets of $\{v_1, v_2, \ldots, v_j\}$ and hit $\mathcal{X}$. Hence, $T[\mathcal{X}, j] \leq min(|S|, |S'| + 1) \implies T[\mathcal{X}, j] \leq \min(T[\mathcal{X}, j-1], 1 + T[\mathcal{X}', j-1])$ which concludes the correctness of the recurrence relation. Hence, the entry $T[\mathcal{H}, n]$ corresponds to the size of a minimum hitting set of $\mathcal{H}$. □

In case of the Set Cover problem, neither do we know any better algorithm nor is there any known reduction that refutes SETH. This has led Cygan et al to propose the Set Cover Conjecture:

Conjecture 3.1 (Set Cover Conjecture [CDL+16])**.** *There is no algorithm for the* Set Cover *problem that runs in* $\mathcal{O}^*((2-\epsilon)^n)$ *for any* $\epsilon > 0$ *where* n *is the size of the universe.*

From the duality between the Set Cover problem and the Hitting Set problem, under the Set Cover Conjecture, the Hitting Set problem does not have any algorithm that runs in $\mathcal{O}^*((2-\epsilon)^m)$ for any $\epsilon > 0$ where m is the size of the family $\mathcal{H}$.

Fixed-parameter tractability. The *intractability* of the general Hitting Set problem with unbounded subset size percolates to parameterized regime as well. Under this assumption, the next theorem shows that Hitting Set is unlikely to have an algorithm of the form $f(k) \cdot n^{\mathcal{O}(1)}$ where f is a computable function and k is the size of the solution H.

Theorem 7. Hitting Set *with solution size as the parameter is* W[2]*-complete.*

Note that W[2]-completeness of the Hitting Set problem is implied indirectly from the work of Paz and Moran [PM81] which predates the parameterized complexity framework. The best known algorithm for Hitting Set can be inferred from the work of Eisenbrand and Grandoni [EG04] that takes time

$n^{k+\mathcal{O}(1/k)}$, which is an improvement over the algorithm that tries all subsets of size at most k and runs in time $\mathcal{O}(n^{k+1})$. W[2]-completeness of Hitting Set with the hitting set as the parameter effectively rules out any kernel for it as well.

Approximation algorithms. If we allow only polynomial time, then one can approximate Set Cover within a factor of $\mathcal{O}(\log n)$ (here n is the size of the universe) [WS11] using a greedy approach: the algorithm picks from the family of unpicked sets, the set with largest number of uncovered elements. The approximation algorithm for Set Cover implies that Hitting Set can be approximated within a factor of $\mathcal{O}(\log m)$. Under different complexity theory conjectures, it has been shown that the above greedy approximation algorithm for Set Cover is essentially tight. Lund and Yannakakis [LY94] proved for the Set Cover problem that for any $\alpha < \frac{1}{4}$, the existence of a polynomial-time $\alpha \ln n$-ratio approximation algorithm would imply that problems in NP has a quasipolynomial, i.e., $n^{\mathcal{O}(poly(\log n))}$ deterministic algorithm. This result was improved to $(1 - o(1)) \log n$ by Feige [Fei98]. A $c \cdot \log n$-approximation under the assumption that P$\neq$ NP was established by Safra and Raz [RS97], where c is a constant. A similar result for larger values of c was proved by Alon, Moshkovitz and Safra [AMS06]. Recently, Moshkovitz has shown $(1-\epsilon) \log n$ inapproximability of Set Cover under P $\neq$ NP [Mos15].

Relaxing the specifications. Considering these theoretical barriers, the general Hitting Set problem seems to be *too* general. Often, the hitting set formulation of problems have a lot of structure. To understand the nature of the intractability of the Hitting Set problem, one needs to probe it with more parameters that reflect some structure in the input. Later we discuss about restrictions of Hitting Set called d-Hitting Set in which we restrict the size of each set in the family to some constant d which is called the d-Hitting Set problem. For example, the Vertex Cover problem can be seen as 2-Hitting Set problem with the vertex set as the universe and the edge set as the family $\mathcal{H}$. The Feedback Vertex (arc) Set in Tournaments [DGH$^+$10], Cluster Vertex Deletion [HKMN10] and Odd Cycle Transversal in perfect graphs [CFK$^+$15] can be formulated as a 3-Hitting Set problem. The Feedback Vertex Set in Bipartite Tournaments [Hsi11] and Cograph Deletion [NG10] can be formulated as a 4-Hitting Set problem while Split Vertex Deletion [CP12] can be formulated as a 5-Hitting Set problem.

3.2 d-Hitting Set

Even d-Hitting Set for constant $d > 1$ remains NP-complete. This follows immediately from NP-completeness of the Vertex Cover problem. Next, we outline some of the progress made for this problem in the paradigm of exact-exponential-time algorithms, parameterized algorithms, approximation algorithms and kernelization.

Exact-exponential-time algorithms. As we saw, under SETH, it is unlikely that there exists an algorithm that solves the general Hitting Set in time $\mathcal{O}((2-\epsilon)^n)$ time for $\epsilon > 0$. Using the iterative compression technique, Fomin et al [FGK+10] showed that if $(d-1)$-Hitting Set can be solved in $\mathcal{O}^*(c^n)$ time for $1 \leq c \leq 2$, then d-Hitting Set can be solved in $\mathcal{O}^*((\frac{1+\sqrt{1+4c}}{2})^n)$ time. For example, the $\mathcal{O}(1.6268^n)$ algorithm for 3-Hitting Set [Wah07] yields an algorithm for 4-Hitting Set running in $\mathcal{O}^*((\frac{1+\sqrt{1+4\times 1.6268}}{2})^n) = \mathcal{O}(1.8704^n)$ time. For values of $c \geq 1.6553$, they provide an alternate algorithm (again based on iterative compression) that performs slightly better than the above algorithm. We remark that these are not the currently fastest known exponential-time algorithms for d-Hitting Set, for the fastest ones see [FGLS16].

Parameterized algorithms. For constant d, d-Hitting Set parameterized by the solution size k is in FPT. A simple $\mathcal{O}^*(d^k)$ time branching algorithm can be obtained as follows: maintain a partial hitting set $S \subseteq U$ and a family of sets $\mathcal{A} \subseteq \mathcal{H}$ that has not been hit so far. Pick a set $F \in \mathcal{A}$ and for each choice of an element $v \in F$ branch with the possibility that $v \in S$. If in any branch $|S| > k$, then terminate that branch. If there exists a branch in which $\mathcal{A} = \emptyset$ and $|S| \leq k$, return *yes*, otherwise return *no*. For d-Hitting Set, Niedermeier and Rossmanith [NR03] presented an $\mathcal{O}(c^k + n)$ time algorithm with $c = d - 1 + \mathcal{O}(d^{-1})$. In the same work, they gave an algorithm running in $\mathcal{O}(2.270^k + n)$ for 3-Hitting Set which was improved by Wahlström [Wah07] to $\mathcal{O}^*(2.076^k)$ time. Next, we show the iterative compression based algorithm for d-Hitting Set due to Fomin et al. [FGK+10].

Theorem 8 ([FGK+10])**.** *Suppose there exists an algorithm to solve* $(d-1)$-Hitting Set *in time* $\mathcal{O}(c^k \cdot n^{\mathcal{O}(1)})$*, where* $c \geq 1$*. Then* d-Hitting Set *can be solved in time* $\mathcal{O}((1+c)^k \cdot n^{\mathcal{O}(1)})$*.*

Proof. The algorithm we consider is based on the iterative compression technique. Starting with an instance of d-Hitting Set, we reduce the problem to solving instances of $(d-1)$-Hitting Set.

Let $I := (U, \mathcal{F}, k)$ be a d-Hitting Set instance. Consider some enumeration of the elements in U i.e. $U = \{u_1, u_2, \ldots, u_n\}$. For each $i \in [n]$, let $U_i = \{u_1, u_2, \ldots, u_i\}$ and let $\mathcal{F}_i$ be the family of sets in $\mathcal{F}$ which are subsets of U_i, i.e. $\mathcal{F}_i = \{X \in \mathcal{F} \mid X \subseteq U_i\}$. Note that $U_n = U$ and $\mathcal{F}_n = \mathcal{F}$. The method of iterative compression is to use a hitting set H_{i-1} of size at most k for d-Hitting Set instance $I_{i-1} := (U_{i-1}, \mathcal{F}_{i-1}, k)$ to obtain a hitting set of H_i size at most k for d-Hitting Set instance $I_i := (U_i, \mathcal{F}_i, k)$.

In the base case assuming $k \geq 1$, if $\{u_1\} \in \mathcal{F}$, then $H_1 = \{u_1\}$. Otherwise $H_1 = \emptyset$. Note that at the ith stage of the iteration, $S_i := H_{i-1} \cup \{u_i\}$ is a hitting set of $\mathcal{F}_i$ and $|H_{i-1}| \leq |H_i| \leq |S_i| = |H_{i-1}| + 1 \leq k + 1$. Next we define a subroutine $\mathcal{B}$ that takes (U_i, F_i, S_i, k) as input and outputs H_i or concludes that I is a no-instance.

The subroutine $\mathcal{B}$ works as follows: For every partition $(Z, \bar{Z})$ of S_i, construct d-Hitting Set instance $I'_i(Z) := (U_i \setminus \bar{Z}, \mathcal{F}'_i, k - |Z|)$ where $\mathcal{F}'_i := \{F \in \mathcal{F} \mid$

$F \cap Z = \emptyset\}$. If $I'_i(Z)$ is a no-instance for every partition $(Z, \bar{Z})$ of S_i, then output 'no' and terminate the algorithm. Otherwise, let A be the solution for I'_i. Output $H_i := Z \cup A$.

If the subroutine $\mathcal{B}$ does not terminate with 'no', the algorithm outputs H_n as the solution.

Correctness: Suppose I_i has a hitting set H_i of size at most k. By guessing all the partitions of S_i, the algorithm guesses $H_i \cap S_i$ and $S_i \setminus H_i$. To that end, for every partition $(Z, \bar{Z})$ of $H_{i-1} \cup \{u_i\}$, we assume that $Z \subseteq H_i$ and $\bar{Z} \cap H_i = \emptyset$. Indeed, if a set $X \in \mathcal{F}_i$ is a subset of $\bar{Z}$, then such a set by the above assumption can not be hit by H_i and hence we are dealing with a *bad* partition $(Z, \bar{Z})$. In this case, we discard this partition and try a different partition of $H_{i-1} \cup \{u_i\}$. If all partitions happen to be bad in this sense, then there does not exist any set H_i of size at most k that hits $\mathcal{F}_i$. Hence, we conclude that I does not have any hitting set of size at most k and we terminate the algorithm. Assuming that $(Z, \bar{Z})$ is not a bad partition, sets in $\mathcal{F}_i$ that have non-empty intersection with Z are *already hit* and hence can be safely removed from the family $\mathcal{F}_i$. Let $\mathcal{F}'_i$ be the remaining sets in $\mathcal{F}_i$. Every set in $\mathcal{F}'_i$ has non-empty intersection with $\bar{Z}$ and since these elements do not belong to H_i, we can safely remove them from each set in $\mathcal{F}'_i$. This process decreases the cardinality of each set in $\mathcal{F}'_i$ by at least one. A hitting set of $(U_i \setminus \bar{Z}, \mathcal{F}'_i)$ of size at most $k - |Z|$ implies that $(U_i, \mathcal{F}_i)$ has a hitting set of size at most k. Hence, $(U_i \setminus \bar{Z}, \mathcal{F}'_i, k - |Z|)$ is a $(d-1)$-Hitting Set instance that can be solved in $\mathcal{O}(c^{k-|Z|} \cdot n^{\mathcal{O}(1)})$ time. Now, we do the above process for every partition of $H_{i-1} \cup \{u_i\}$. Hence, the overall running time in given by $\sum_{i=0}^{k} \binom{k+1}{i} c^{k-i} \leq 2 \cdot (1+c)^k$ by binomial theorem. □

As an immediate consequence of the above theorem is that d-Hitting Set can be solved in $(c_3 + d - 3)^k \cdot n^{\mathcal{O}(1)}$ time where c_3 is such that 3-Hitting Set can be solved in $c_3^k \cdot n^{\mathcal{O}(1)}$ time. By Wahlström's algorithm [Wah07], $c_3 \leq 2.0755$. Hence, d-Hitting Set can be solved in $(d - 0.9245)^k \cdot n^{\mathcal{O}(1)}$ time. For small values of d, this algorithm is faster than the algorithm due to Niedermeier and Rossmanith.

Approximation algorithms. One can approximate d-Hitting Set within a factor of d [Hoc82]. Under the Unique Games Conjecture, the above factor can not be improved in polynomial time [KR08]. A hypergraph $\mathcal{G}$ is a generalization of graph. Here the edge set is a family of subsets of the vertex sets while in graphs the edge set is a family of subsets of size exactly 2 of the vertex set. In this sense, hypergraphs provide a natural link to hitting sets. Hitting Set problem can be formulated as Vertex Cover in a hypergraph $\mathcal{G}$ in which the vertex set is the universe U and the edge set is the family $\mathcal{H}$. A vertex cover H of $\mathcal{G}$ is a set of vertices such that every edge is incident at at least one vertex in H which is a hitting set for $\mathcal{H}$. When d-Hitting Set is formulated as a hypergraph vertex cover problem, then d becomes the maximum size of any edge and the maximum degree of any vertex f is the maximum frequency of an element. In the equivalent Set Cover instance, f becomes the maximum size of any set and d becomes the frequency. The greedy algorithm of the Set Cover actually provides

$\log |S_{max}|$ factor approximation where S_{max} is the set of maximum cardinality in the family [WS11]. Hence, we get a $\log f$ factor approximation for the HITTING SET.

Kernelization. With respect to polynomial-time kernelization algorithms, Dell and van Melkebeek [DvM14a] showed that for any $d \geq 2$, the d-HITTING SET problem parameterized by $|U|$ does not have a compression with bitsize $\mathcal{O}(|U|^{d-\epsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Intuitively, by a compression of a parameterized problem, we mean a polynomial-time algorithm that as an input an instance of a parameterized problem and outputs an equivalent instance of another (may be unparameterized) problem. Furthermore, the size of output (also called the bitsize of the compression) instance is bounded by some function of the input parameter k alone. The difference between kernelization and compression is that a kernelization algorithm outputs an equivalent instance of the same parameterized problem as the input problem while compression can output an instance of a different problem.

If d is not a constant, d-HITTING SET is equivalent to the general HITTING SET which is $\mathsf{W[2]}$-hard and have no kernel of any size unless $\mathsf{FPT} = \mathsf{W[2]}$. But, when parameterized by d and k, the Sunflower lemma yields a kernel for d-HITTING SET with at most $d!k^d$ sets and at most $d!k^d \cdot d^2$ elements [FG06]. Using Crown decomposition, Abu-Khazam [AK10] improved this to a kernel with at most $(2d-1)k^{d-1} + k$ elements. As an example of the Sunflower lemma technique for obtaining kernels, we state a kernel for d-HITTING SET and sketch its proof (see [CFK+15] for a complete proof). The Sunflower lemma was first proved by Erdős and Rado (1960).

Definition 3.2 (Sunflower)**.** A *sunflower* with k *petals* and a *core* Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$: the sets $S_i \setminus Y$ are petals and we require *none of them to be empty.*

Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

Theorem 9 (Sunflower Lemma (Erdős, Rado))**.** *Let $\mathcal{H}$ be a family of sets (without duplicates) over a universe U, such that each set in $\mathcal{H}$ has cardinality exactly d. If $|\mathcal{H}| > d!(k-1)^d$, then $\mathcal{H}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{H}|$, U and k.*

Theorem 10. *d-HITTING SET admits a kernel with at most $d!k^d$ sets and at most $d!k^d \cdot d^2$ elements.*

Proof Sketch: The usual approach of using the Sunflower lemma is to show that if the size of the family is sufficiently large, then there exists a sunflower that would force some of the elements into the solution. For the problem at hand, if there exists a sunflower with at least $k+1$ petals, then any hitting set of size at most k must intersect the core of the sunflower, otherwise any hitting set will be forced to contain at least $k+1$ distinct elements belonging to the $k+1$ petals (minus the core). This observation yields a simple reduction rule: if $\mathcal{H}$ contains a

sunflower with at least $k+1$ petals with core Y, then run the algorithm again after removing from $\mathcal{H}$ all the sets belonging to the sunflower and including Y as a new member in the family. At the same time update the universe to only contain elements that belong to some set in the updated family. The parameter k does not change. The number of sets and the size of the universe in the reduced instance follows from the statement of the Sunflower lemma. □

Chapter 4

Implicit Hitting Set

In the previous chapter, we saw algorithms and results for problems that can be formulated as an explicit Hitting Set problem. In this chapter, we study some optimizations problems that are not usually stated as a Hitting Set problem explicitly, but the problems can be stated as a hitting set problems implicitly. Therefore the techniques that are useful for Hitting Set could also useful in this setting. We start with problems with broad classification based on forbidden sets and VC-dimension and narrow down to specific problems like Feedback Vertex Set and Dominating Set and finally discuss about separation oracles.

4.1 Hereditary Graph properties

A *graph property* is a set Π of graphs and any graph in Π is said to be a Π-graph. Connected graphs, bipartite graphs, planar graphs, triangle-free graphs, perfect graphs etc. are the examples of graph properties. If for any graph $G \in \Pi$, every induced subgraph of G is also a Π-graph, then Π is a *hereditary property*. Except for connected graphs, every other graph property stated above is a hereditary property. A property Π has a *forbidden set characterization* if there is a set $\mathcal{F}$ of graphs such that a graph is a Π-graph if and only if it does not contain any graph in $\mathcal{F}$ as an induced subgraph, and it has a *finite forbidden set characterization* if $\mathcal{F}$ is a finite set. Every hereditary property has a forbidden set characterization. For example, cluster (P_3), cographs (P_4), triangle-free (K_3), co-trivially perfect ($2K_2, P_4$), split graphs ($2K_2, C_4, C_5$), deletion to chain ($2K_2, K_3, C_5$), threshold ($2K_2, C_4, P_4$) etc. admit finite forbidden set characterizations. Here P_i, C_i, and K_i is a path, cycle and complete graph on i vertices, respectively. Each property Π naturally defines a set of *graph modification* problems where the input is a graph G and the task is to *modify* G as little as possible to obtain a graph in Π. By modification, we mean adding and deleting edges and vertices.

$\Pi(i,j,k)$-Graph modification Problem

Input:	A graph G and positive integers i, j, k
Question:	Find a set of vertices $S \subseteq V(G)$, a set of edges $E_1 \subseteq E(G)$ and a set of edges E_2 in the complement graph of G with $\|S\| \leq i, \|E_1\| \leq j$ and $\|E_2\| \leq k$ such that the graph $G - S - E_1 + E_2$ is a Π-graph

When such subsets exist, G is called a $\Pi(i,j,k)$-graph and the set $S \cup E_1 \cup E_2$ is referred to as a *modifier* of G.

Note that if j and k are 0, we have the Vertex Deletion problem. If i is 0, we have Editing problem. If i and j are 0, the problems are called Completion problems, while if i and k are 0, we have Edge Deletion problems. Note that Completion problems and Deletion problems are related in that the Deletion problem on G to a hereditary property (characterized by forbidden sets $\{F_1, F_2, \dots\}$) is equivalent to the Completion problem on the complement $\bar{G}$ to the hereditary property characterized by the forbidden set $\{\bar{F_1}, \bar{F_2}, \dots\}$ i.e. add edges so that (say) $\bar{F_1}$ is not an induced subgraph of $\bar{G}$. Some of the well-studied examples of Editing problems are Split Editing [HS81], Cluster Editing [KM86] and Chordal Editing [DDLS15]. Naturally, for each Editing problem, there are corresponding Deletion and Completion problems.

If i, j and k are considered constants that are not part of the input, then the above problem is trivially polynomial-time solvable by the exhaustive search method for any polynomial-time recognizable property Π. On the other hand, the problem is NP-hard for any non-trivial hereditary property Π, if the parameter (i, j, k) are part of the input. For many non-trivial properties, the Vertex Deletion problem was shown to be NP-complete by Krishnamoorthy and Deo [KD79]. Later, these results were extended by Lewis and Yannakakis to hereditary properties [LY80a]. We call a property *non-trivial* if it is true for infinitely many graphs and false for infinitely many graphs. Cai [Cai96] studied the fixed-parameter tractability of graph modification problems.

Theorem 11 ([Cai96])**.** *The $\Pi(i,j,k)$-graph modification problem parameterized by $i + j + k$ is fixed-parameter tractable for any hereditary property Π that admits a finite forbidden set characterization.*

Exact Exponential-time algorithms By adapting the original NP-hardness construction Komusiewicz showed that under the Exponential Time Hypothesis tight complexity results can be obtained [Kom15]. The simple try-all-subsets algorithm for Π-Vertex Deletion is essentially tight i.e. it does not admit a $2^{o(n)}$-time algorithm where n is the number of vertices in G. If Π contains all independent sets, then there is no $2^{o(n+m)}$-time algorithm for Π-Vertex Deletion where m is the number of edges in the graph. If there is a fixed independent set that is not contained in Π and containment in Π can be determined in $2^{\mathcal{O}(n)}$ time or $2^{o(m)}$ time, then Π-Vertex Deletion can be solved in $2^{\mathcal{O}(\sqrt{m})} + O(n)$ or

$2^{o(m)} + \mathcal{O}(n)$ time, respectively. In addition, Komusiewicz showed that Π-Vertex Deletion can not be solved in $2^{o(\sqrt{n})}$ time if G is planar and Π is hereditary and contains and excludes infinitely many planar graphs.

Connection to d-Hitting Set For the Vertex Deletion problem for a hereditary property Π, one may list all the occurrences of graphs in the family $\mathcal{F}$ of forbidden graphs corresponding to Π in the given graph G to obtain a family of sets $\mathcal{H}$. Then, deleting k vertices from G to get a Π-graph reduces to hitting $\mathcal{H}$ by a set of size at most k. This method would be practical only when $\mathcal{F}$ is finite as the sets in $\mathcal{H}$ are bounded in size by the size of the largest set in $\mathcal{F}$ (say d). In this case the formulated problem is exactly the d-Hitting Set problem.

In case of Vertex Deletion problems for hereditary properties with finite forbidden set characterization, one can obtain a kernel of size $\mathcal{O}(k^d)$. For Edge Deletion problems, polynomial kernel is unlikely in general as Guillemot et al [GPP10] showed that P_l-free Edge Deletion ($l \geq 12$) and C_l-free Edge Deletion ($l \geq 13$) problems do not admit polynomial kernels unless NP $\subseteq$ coNP/poly. Also see [KW13, CC15]. In the case of edge deletion, deleting edges can create new forbidden subgraphs that weren't there before as the closure under induced subgraph property is lost. For example, Claw-free graphs do not contain $K_{1,3}$ as an induced subgraph. In the Claw-Free Deletion problem, the task is to delete vertices or edges to reduce the input graph to a claw-free graph. While the vertex version has a polynomial kernel, it is open whether there exists a polynomial kernel for the edge version of the problem.

A dual of Vertex Deletion problem Raman and Khot [KR02] addressed the parametric dual of the node deletion problem defined below. Given any property Π, the problem $P(G, k, \Pi)$ is defined as follows.

$P(G, k, \Pi)$

Input:	A graph G on n vertices and positive integer $k \leq n$
Question:	Is there a subset $S \subseteq V(G)$ with $\|S\| = k$ such that the subgraph of G induced by S, $G[S]$ is in Π?

This problem is the same as $\Pi(n-k, 0, 0)$ problem (i.e. can we delete all but k vertices of G to get a graph in property Π) and hence NP-hard. However, the parameterized complexity of this problem does not follow from the complexity of the problem addressed by Cai even for properties having a finite forbidden set. This is because the NP-hard reduction reduces a general instance (G, i, j, k) of the $\Pi(i, j, k)$ problem to the instance $P(G, n-k, \Pi)$. This is not a parameterized reduction as the parameter of the reduced instance can be a function only of the original parameter k (and not of n) in a parameterized reduction.

Raman and Khot [KR02] proved that if Π includes all trivial graphs (a graph with no edges) but not all complete graphs, or vice versa, then the problem $P(G, k, \Pi)$ is W[1]-complete. The proof is by a parameter preserving reduction

from the Independent Set problem. If Π includes all trivial graphs and all complete graphs, or excludes some trivial graph and some complete graph, then the problem is fixed parameter tractable.

4.2 Algebraic problems

Now instead of using a structural route to define $\mathcal{F}$ as was done in the previous section, in this section, we take an algebraic route whose study was initiated by Meesum et al for undirected graphs in [MMS15] and directed graphs in [MS16]. More formally, given a fixed positive integer r, we define $\mathcal{F}_r$ as the family of graphs where for each $G \in \mathcal{F}_r$, the rank of the adjacency matrix A_G of G (denoted by $rank(A_G)$) is at most r.

r-Rank Vertex Deletion

Input:	A graph G and positive integers k
Question:	Can we delete at most k vertices from G so that $rank(A_G) \leq r$

Similarly, the edge deletion and editing variants can be defined. These problems generalize the Vertex Cover problem and a variant of the d-Cluster Editing problem. These problems are related to some well known problems in graph algorithms. Observe that if $rank(A_G) = 0$, then G is an empty graph and if $rank(A_G) = 2$ then G is a complete bipartite graph with some isolated vertices. There are no graphs such that $rank(A_G) = 1$. So for $r = 0$, r-Rank Vertex Deletion is the well known Vertex Cover problem. Similarly for $r = 2$, a solution to r-Rank Edge Deletion is a complement of a solution to Maximum Edge Biclique, where the goal is to find a bi-clique (complete bipartite graph) subgraph of the given graph with maximum number of edges.

Theorem 12 ([MMS15]). r-Rank Vertex Deletion *is* NP*-complete.*

We can formulate r-Rank Vertex Deletion as a Hitting Set problem. Let G be a graph with adjacency matrix $A_{n\times n}$. Let $\mathcal{H}(G) = \{X \cup Y \mid X, Y \subseteq V(G), |X| = |Y| = r+1, rank(A_{X,Y}) = r+1\}$. It is easy to verify that for any $H \subseteq V(G)$, the rank of the adjacency matrix of $G \setminus H$ is at most r if and only if H is a hitting set of $\mathcal{H}(G)$ [MMS15].

Theorem 13 ([MMS15]). r-Rank Vertex Deletion *admits an* FPT *algorithm running in time* $2^{O(k \log r)} n^{O(1)}$.

Theorem 14 ([MMS15]). r-Rank Vertex Deletion *admits a kernel having at most* $2(r+1) \cdot (2(r+1))!(k+1)^{2r+2}$ *vertices.*

4.3 VC-dimension

Another way to put restrictions on sets in $\mathcal{H}$ is to consider the VC-dimension of $\mathcal{H}$. VC-dimension is a property of set families, where set families with low VC-dimension are structured is a certain sense. To define the VC-dimension of $\mathcal{H}$, let

us first define *restriction* of a family of sets to a set. For any set $A \subseteq U$ and a family of subsets $\mathcal{H}$ of U, the restriction of $\mathcal{H}$ to A is define as $\mathcal{H}_A = \{F \cap A \mid F \in \mathcal{H}\}$. A set A is said to be *shattered* by $\mathcal{H}$ if $\mathcal{H}_A = 2^A$, i.e. the set of all subsets of A. The *Vapnik-Chervonenkis dimension* (or VC-dimension) of $\mathcal{H}$ denoted by $VC(\mathcal{H})$, is the cardinality of the largest set shattered by $\mathcal{H}$. If there is no such largest set, we say that $VC(\mathcal{H})$ is infinity.

It is immediate that if every set in $\mathcal{H}$ has cardinality at most d, then the VC-dimension of $\mathcal{H}$ can be at most d. If fact, it can be shown that the VC-dimension of a family of sets $\mathcal{H}$ over a universe U of cardinality n is at most $\log n$. Consequently computing the VC-dimension of a family of sets $\mathcal{H}$ is unlikely to be NP-complete [PY96]. Downey and Fellows showed that deciding whether $\mathcal{H}$ has VC-dimension at most k is W[1]-complete [DF95]. Bronnimann and Goodrich [BG95] gave an almost optimal ($O(\log k)$ where k is the size of the optimal solution) approximation algorithm for Hitting Set on families with bounded VC-dimension. This was later improved by Evan et al. [ERS05] and by Agarwal and Pan [AP14].

Unfortunately, VC-dimension can not be used to separate problems are FPT from the problems that are W[1]-hard. There are some FPT classes that have unbounded VC-dimension, while W[1]-hard classes with VC-dimension 3 are known. Bringmann et al [BKMN16] proved that Hitting Set restricted to VC-dimension 2 is W[1]-hard. Moreover, if Hitting Set problem restricted to VC-dimension 2 can be solved in time $f(k) \cdot |U|^{o(k/\log k)}$, where f is an arbitrary function and k is the solution size, then ETH fails.

In geometric examples of Hitting Set, the input set system is defined by the incidences between (typically) low complexity geometric shapes, such as points, intervals, lines, disks, rectangles, hyperplanes, etc. [VC71, BEHW89, HKL$^+$13]. The complexity of these problems varies widely from being polynomial to W[1]-hard. Now we state some of the known results about these problems without definition and the reader to directed to appropriate papers for more detail. Line intervals is polynomial-time solvable whereas Disjoint Rectangle Stabbing [HKL$^+$13] and Pseudoline arrangement are NP-complete and in FPT. The input set systems for each of these problems have VC-dimension 2. Similar story plays for VC-dimension 3: Halfplane arrangement in $\mathbb{R}^2$ [HL12] is polynomial-time solvable while Collection of unit disks and squares in $\mathbb{R}^2$ [GKW08] remain W[1]-hard. Halfspace arrangement in $\mathbb{R}^3$ [BKMN16] with VC-dimension 4 is W[1]-hard whereas Hyperplane arrangement in $\mathbb{R}^d$ with VC-dimension $d+1$ is FPT.

4.4 Dominating Set

Among standard graph problems, Dominating Set comes closest to being a Hitting Set problem. In fact, a variant of Dominating Set called Red-Blue Dominating Set is equivalent to Hitting Set. In the Red-Blue Dominating Set problem, the input is a bipartite graph (R, B, E) and an integer k and the task is to determine whether there exists a subset $H \subseteq R$ of size at most k such

that every vertex in B has at least one neighbor in H. H is called a red-rlue dominating set. Given a HITTING SET instance $(U, \mathcal{H}, k)$, we can obtain a RED-BLUE DOMINATING SET instance as follows: define $R := U$. For each set $X \in \mathcal{H}$, include a vertex v_X in B. For every vertex $v_X \in B$, v_X is adjacent to every element in $X \subseteq U$. It is easy to verify that (R, B, E) has a red-blue dominating set of size at most k if and only if $\mathcal{H}$ has a hitting set of size at most k. Observe that the above transformation is invertible, i.e. given a RED-BLUE DOMINATING SET instance, we can construct an equivalent HITTING SET instance in which the family of sets is the family of neighborhoods of vertices in B.

DOMINATING SET is NP-hard even on very restricted graph classes, e.g. planar graphs of bounded degree [GJ79a]. Naturally, the complexity of the problems has been extensively studied under different algorithmic frameworks, notably approximation algorithms and parameterized complexity. Using the approximation algorithm for HITTING SET, we can obtain an $\mathcal{O}(\log n)$ approximation for DOMINATING SET. On general graphs, the problem is $(1 - \epsilon) \log n$ hard to approximate for any $\epsilon > 0$ under standard complexity theoretic assumptions [CC04]. But, DOMINATING SET admits a *polynomial-time approximation scheme* (PTAS) for planar graphs, H-minor-free graphs, unit disk graphs and growth-bounded graphs [IMR$^+$98, NHK08].

In parameterized complexity, DOMINATING SET is a standard example of a W[2]-complete problem. This excludes the possibility of having FPT algorithms, thereby implying no kernels for the problem unless W[2] = FPT. Therefore, attempts have been made to find graph classes on which the results are not so discouraging. This line of research has lead to very fruitful and insightful discoveries in parameterized complexity in general and in kernelization in particular.

Intuitively, DOMINATING SET is hard because it places very few restrictions on the input. Whenever the set-family incidence matrix is symmetric, the problem can be formulated as a DOMINATING SET instance. Special cases where DOMINATING SET is FPT include *biclique-free* graphs [PRS09, TV12] (a family that contains *bounded genus, planar, bounded treewidth* and many other natural classes), *claw-free* graphs [HMvLW11], and graphs *with girth at least five* [RS08]. The structure that makes these special cases of DOMINATING SET tractable can be described in terms of *forbidden patterns* in the adjacency matrix of G. For instance, biclique-freeness simply translates to the avoidance of all-1s submatrix of a certain size.

In the regime of kernelization, for DOMINATING SET, it all started with the linear kernel on planar graphs of Alber et al. [AFN04]. Next, a polynomial kernel on H-topological-minor free graphs was given by Alon and Gutner [AG08]. More recent efforts have lead to linear kernels on bounded genus graphs [BFL$^+$09], apex-minor-free graphs [FLST10], H-minor-free graphs [FLST12], and H-topological-minor-free graphs [FLST13]. Philip et al. [PRS12] obtained a kernel of size $\mathcal{O}(k^{(d+1)^2})$ on d-degenerate graphs, for constant d, and more generally a kernel of size $\mathcal{O}(k^{\max(i^2,j^2)})$ on graphs excluding the complete bipartite graph $K_{i,j}$ as a subgraph. Drange et al. [DDF$^+$16] provided a linear kernel for graphs of bounded expansion and an $\mathcal{O}(k^{1+\delta})$-size kernel for nowhere-dense graphs, for every constant $\delta > 0$ (See Figure 4.1). On the lower bounds side, Cygan et al. [CGH13] have

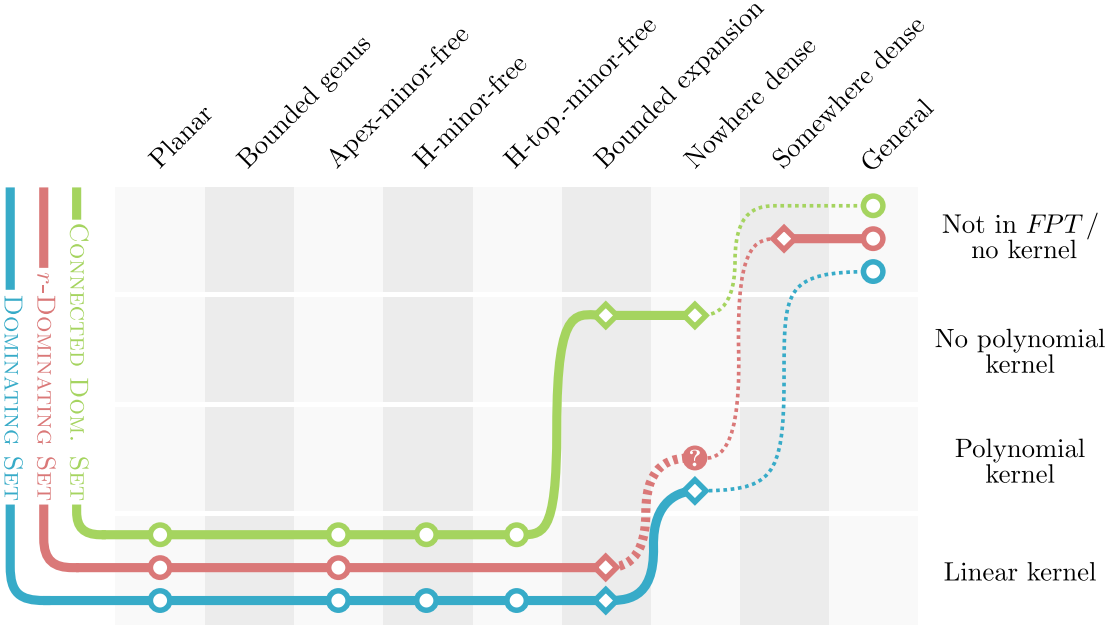

Figure 4.1: Kernels for various graph classes for the Dominating Set problems. Figure taken from [DDF+16]

shown that existence of an $\mathcal{O}(k^{(d-1)(d-3)-\epsilon})$ kernel, $\epsilon > 0$, for Dominating Set on d-degenerate graphs would imply NP $\subseteq$ coNP/poly . In Chapter 8 we study about a variant called Connected Dominating Set problem.

4.5 Vertex Cover

Vertex Cover is one of the most studied graph problems. Vertex Cover is precisely the 2-Hitting Set problem with U being the vertices in the graph and $\mathcal{H}$ is the set of edges in the graph. The Vertex Cover problem is NP-complete [GJ79a] and is often used as a starting point for NP-hardness reductions. It remains NP-complete even in cubic graphs and even in planar graphs of degree at most 3.

We already saw a 2-approximation algorithm in Chapter 1. The Vertex Cover problem is APX-complete, i.e. there exists an $\epsilon > 0$ such that a $(1+\epsilon)$-approximation would imply that P = NP. Using PCP theorem, Dinur and Safra [ID05] proved that Vertex Cover cannot be approximated within a factor of 1.3606 unless P = NP. Assuming the Unique Games Conjecture, Vertex Cover cannot be approximated within any constant factor better than 2 [KR08].

As we saw, there is a simple $2^k \cdot n^{\mathcal{O}(1)}$ algorithm for Vertex Cover. After a series of improvements in the base of the exponent, the current best algorithm runs in $\mathcal{O}(1.2738^k + k \cdot n)$ [CKX10]. Under ETH, the running time cannot be improved to $2^{o(k)}$. The kernel with $2k$ vertices for Vertex Cover is considered one of the fundamental results in the field of kernelization. Dell and van Melkebeek [DvM14a] showed that this kernel is essentially tight unless NP $\subseteq$ coNP/poly. In particular,

there is no algorithm that reduces all instances (G, k) of Vertex Cover to instances with bitsize $\mathcal{O}(k^{2-\epsilon})$.

4.6 Feedback Vertex Set

A *feedback vertex set* in a graph G is a vertx set S such that $G - S$ is acyclic. For undirected graphs this means that $G - S$ is a forest, while for directed graphs this implies that $G - S$ is a directed acyclic graph (DAG). In the Feedback Vertex Set (FVS) problem we are given as input an *undirected* graph G and integer k, and asked whether there exists a feedback vertex set of size at most k. The corresponding problem for directed graphs is called Directed Feedback Vertex Set (DFVS). We have already described how the Feedback Vertex Set can be seen as an implicit Hitting Set problem in the beginning of Chapter 3. Both versions of FVS are NP-complete [GJ79b] and have been extensively studied from the perspective of approximation algorithms [BBF99, ENSS98], parameterized algorithms [CLL+08, CNP+11, KP14], exact exponential time algrithms [Raz07, XN15] as well as graph theory [RRST96].

Karp's proof of NP-hardness also implies that the problem on undirected graphs is APX-hard. The best known approximation algorithm on undirected graphs achieves a factor of two [BBF99]. The Vertex Cover problem is reducible to Feedback Vertex Set problem in an approximation preserving manner [LY80a], so that any factor c approximation algorithm for the Feedback Vertex Cover implies a factor c approximation algorithm for Vertex Cover. This implies that under Unique Games Conjecture, the algorithm of Bafna et al. [BBF99] is the best possible. The best known approximation factor in polynomial time for Directed Feedback Vertex Set is $\mathcal{O}(\log n \log \log n)$ [ENSS98, Sey95] and the problem does *not* admit a constant factor approximation assuming the Unique Games Conjecture [GHM+11].

Razgon [Raz06] gave a 1.8899^n-time algorithm for finding a minimum feedback vertex set in an undirected graph, which was the first exact algorithm for the problem that broke the trivial barrier of 2^n. Later, Fomin et al. [FGPR08] improved the result to $1.7548^n \cdot n^{\mathcal{O}(1)}$ which was further improved by Xiao and Nagamochi [XN15] to $1.7356^n \cdot n^{\mathcal{O}(1)}$. Note that using the $3.593^k \cdot n^{\mathcal{O}(1)}$ time parameterized algorithm, Fomin et al [FGLS16] have improved the running time to 1.7217^n. Razgon [Raz07] also broke the barrier 2^n for Directed Feedback Vertex Set by giving a $1.9977^n \cdot n^{\mathcal{O}(1)}$-time algorithm.

The Feedback Vertex Set problem in undirected graphs is fixed-parameter-tractable and the current best deterministic algorithm runs in $3.592^k \cdot n^{\mathcal{O}(1)}$ [KP14] which has followed a series of improvements [RSS06, GGH+06, DFL+05, CFL+08, CCL10]. Cygan et al [CNP+11] obtained a randomized algorithm for Feedback Vertex Set running in time $3^k \cdot n^{\mathcal{O}(1)}$. Similarly, it was open for a long time whether Directed Feedback Vertex Set admits an FPT algorithm, that is an algorithm that determines whether there exists a solution of size at most k in time $f(k)n^{O(1)}$. In 2008, Chen et al. [CLL+08] gave an algorithm with running

time $O(4^k k^{O(1)} k! nm)$, and it is a major open problem whether there exists an algorithm with running time $2^{O(k)} n^{O(1)}$.

Burrage et al [BEF+06] gave a kernel on $\mathcal{O}(k^{11})$ vertices for FVS in undirected graphs which was improved to $\mathcal{O}(k^3)$ by Bodlaender et al [BvD10]. Currently, the best known kernel is due to Thomasse [Tho10] with $4k^2$ vertices. There is a polynomial-parameter-transformation from Vertex Cover to Feedback Vertex Set which implies that the above kernel is essentially tight unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Whether a polynomial kernel for Directed Feedback Vertex Set exists is a major open problem in the field.

4.7 Separation Oracles

Often it happens that the formulation of a combinatorial problem as a Hitting Set problem requires an exponential number of subsets to be hit. In such a scenario, listing the family of sets $\mathcal{H}$ explicitly is impractical. It has been observed that in many cases there exist efficient procedures to verify whether a candidate set H is a hitting set and if not, output as subset $X \in \mathcal{H}$ that is not hit. Such procedures are called an *oracle*.

Formally, in an Implicit Hitting Set problem, the input is a universe U and a polynomial-time *oracle* that, given a set H, either determines that H is a hitting set or returns a subset that is not hit by H. Thus, the collection $\mathcal{H}$ of subsets to be hit is not specified explicitly. The objective is to find a small hitting set by making at most $|U|^{\mathcal{O}(1)}$ queries to the oracle. A natural way to use the oracle is: first (1) propose a candidate hitting set H, then (2) use the oracle to check if the candidate set hits all the subsets, and if not obtain a subset X that has not been hit, and finally (3) refine H based on X and repeat until a hitting set is found. Some examples of Implicit Hitting Set problems are Feedback Vertex Set in which implicitly $\mathcal{H}$ is the set of cycles in the input graph.

Linear Programming Separation oracles can be viewed more generally, where an algorithm has to find a solution that satisfies a set of constraints. The set of constraints might not be given explicitly, but rather in the form of an oracle. The algorithm asks the oracle whether the solution is feasible, and the oracle either replies yes or returns a constraint that is violated by the solution. A setting in which this viewpoint is particularly fruitful, is in the context of Linear Programming [GLS81, GLS88]. A Linear Program is an optimization problem where the input is an n-dimensional real vector c, an $m \times n$ matrix A and a vector b and the task is to find an n dimensional real vector x such that $c \cdot x$ is maximized such that $Ax \leq b$. This famously has a polynomial-time algorithm, i.e. an algorithm that finds the best x in time polynomial in n, m and the number of bits in all of the numbers in A, c and b. In fact, the algorithm is polynomial in just n and the bit size (i.e logarithm) of the largest number in the matrices A, c and b. However, how is that even possible: if the running time is say n^2 and m is n^{10}. How can the running time of the algorithm be less than the reading the

entire input? This is where the separation oracle comes in–the $poly(n)$ rather than $poly(input\ size)$ result is in a model where the algorithm that solves the LINEAR PROGRAM *does not* know the matrix A and the vector b! In fact, it only knows c, and all it has access to is a separation oracle. This is an oracle that given an n dimensional vector x either correctly tells us that $Ax \leq b$ or gives us one row that is violated. What the LINEAR PROGRAM algorithm promises is that it can find the optimum x with poly(n) calls to the separation oracle. So if the separation oracle takes time T then the total running time becomes $T \cdot n^{\mathcal{O}(1)}$. In Chapter 7, we'll see an application of a LINEAR PROGRAM and a separation oracle for a linear programming relaxation of ℓ-COMPONENT ORDER CONNECTIVITY problem.

Part II

New Results

Chapter 5

Feedback Vertex Set in Tournaments

5.1 Introduction

In this chapter we consider a restriction of Directed Feedback Vertex Set (DFVS), namely the Feedback Vertex Set in Tournaments (TFVS) problem, from the perspective of parameterized algorithms and exact exponential time algorithms. A *tournament* is a directed graph T such that every pair of vertices is connected by an arc, and TFVS is simply DFVS when the input graph is required to be a tournament. Even this restricted variant of DFVS has applications in voting systems and rank aggregation [DGH+10], and is quite well-studied [CDZ00, DGH+10, GM13, RS06].

The Feedback Vertex Set (FVS) problem remains NP-complete and APX-hard in tournaments. Moreover, Speckenmeyer [Spe89] gave an approximation-ratio preserving polynomial time reduction from the Vertex Cover problem in general undirected graphs to the TFVS. Consequently, the TFVS cannot be approximated in polynomial time within a factor better than 1.3606 unless P=NP [ID05], and not within a factor better than 2 assuming the Unique Games Conjecture (UGC) [KR08]. On the upper bound side, TFVS admits as easy 3-approximation algorithm: while the tournament contains a directed triangle, place all the triangles in the FVS and remove them from the tournament. Cai, Deng and Zang [Cai] improved the simple algorithm and gave a polynomial time algorithm with approximation guarantee 5/2, even in the case when vertices have non-negative weights and one seeks a solution of approximate minimum weight. Mnich et el [MWV16] developed a better, 7/3-approximation algorithm for the minimum weight Feedback Vertex Set problem in tournaments, narrowing the gap to the UGC-based lower bound of 2 to 1/3.

TFVS was shown to be fixed parameter tractable by Raman and Saurabh [RS06], who obtained an algorithm with running time $O(2.42^k \cdot n^{O(1)})$. In 2006, Dom et al. [DGH+06] (see also [DGH+10]) gave an algorithm for TFVS with running time $2^k n^{O(1)}$. Gaspers and Mnich [GM13] provided an exact exponential time algorithm

for TFVS that takes $O(1.674^n)$ time. It is worth noting that this algorithm also lists all inclusion minimal feedback vertex sets in the input tournament. In this chapter we describe a single algorithm whose running time is upper bounded by $O(1.618^k + n^{O(1)})$ and by $O(1.46^n)$. This algorithm crucially depends on a balanced edge partition theorem for general undirected graphs. Essentially this theorem states that the vertices of any undirected m-edge graph G of maximum degree d can be colored white or black in such a way that for each of the two colors, the number of edges with both endpoints of that color is between $m/4 - d/2$ and $m/4+d/2$. This partition theorem is of independent interest, and we believe it will find further applications both in algorithms and in graph theory.

Overview of the algorithm. Let us refer the algorithm by $\mathcal{A}$. As a preliminary step $\mathcal{A}$ applies the kernel of Dom et al. [DGH$^+$10] to ensure that the number of vertices in the input tournament is upper bounded by $O(k^3)$. After this step, $\mathcal{A}$ proceeds in three phases.

In the first phase $\mathcal{A}$ finds, in subexponential time, a "large enough" set M of vertices *disjoint* from the solution H sought for, such that M is evenly distributed in the topological ordering of $T - H$. From the set M it is possible to infer a rough sketch of the unique topological ordering of $T - H$ without knowing the solution H. More concretely, every vertex v gets a tentative position in the ordering such that if v is not deleted, then v's position in the topological order of $T - H$ is close to this tentative position.

This tentative ordering can be used to identify *conflicts* between two vertices u and v. Two vertices u and v are in conflict if their tentative positions are so far apart that it fixes the order in which they have to appear in the topological sort of $T - H$, but the arc between u and v goes in the opposite direction. Thus, if u and v are in conflict then at least one of them has to be in the solution feedback vertex set H.

The second phase of $\mathcal{A}$ eliminates vertices that are in conflict with more than one other vertex. Suppose that u is in conflict with both v and w. If u is not deleted then both v and w have to be deleted. $\mathcal{A}$ finds the optimal solution by branching and recursively solving the instance where u is deleted, and the instance where u is not deleted but both v and w are deleted. This branching step is the bottleneck of the algorithm and gives rise to the $O(1.618^k n^{O(1)})$ and the $O(1.46^n)$ running time bounds.

The third and last phase of $\mathcal{A}$ deals with the case where every vertex has at most one conflict. Here a divide and conquer approach based on a partitioning theorem is applied.

Organization of the chapter. In Section 6.2 we set up definitions and notation, and state a few useful preliminary results. Section 6.4 describes and analyzes the first phase of the algorithm. Section 5.4 contains the second phase, as well as the final analysis of the correctness and running time of the entire algorithm, conditioned on the correctness and running time bound of the third and last phase. In Section 5.5 we formally state and prove our a decomposition theorem for undirected graphs, while the description and analysis of the third phase of the

algorithm is deferred to Section 5.6.

5.2 Preliminaries

We assume that graphs do not contain any self loops. A *multigraph* is a graph that may contain more than one edge between the same pair of vertices. A graph is *mixed* if it can contain both directed and undirected edges. We will be working with mixed multigraphs; graphs that contain both directed and undirected edges, and where two vertices may have several edges between them. A **triangle** in a directed graph is a directed cycle of length 3. A **topological sort** of a directed graph D is a permutation $\pi : V(D) \mapsto [n]$ of the vertices of the graph such that for all edges $uv \in E(D)$, $\pi(u) < \pi(v)$. Such a permutation exists for a directed graph if and only if the directed graph is acyclic. For an acyclic tournament, the topological sort is unique.

When working with a mixed multigraph G we use $V(G)$ to denote the vertex set, $E(G)$ to denote the set of directed edges, and $\mathcal{E}(G)$ to denote the set of undirected edges of G. A directed edge from u to v is denoted by uv. A *supertournament* is a directed graph T such that for every pair of vertices u, v at least one (and possibly both) edges uv and vu are edges of T. Thus, every tournament is a supertorunament, but not vice versa.

Preliminary Results. If a tournament is acyclic then it does not contain any triangles. It is a well-known and basic fact that the converse is also true, see e.g. [DGH$^+$10].

Lemma 5.1. *[DGH$^+$10] A tournament is acyclic if and only if it contains no triangles.*

Lemma 5.1 immediately gives rise to a folklore greedy 3-approximation algorithm for TFVS: as long as T contains a triangle, delete all the vertices in this triangle.

Lemma 5.2 (folklore)**.** *There is a polynomial time algorithm that given as input a tournament T and integer k, either correctly concludes that T has no feedback vertex set of size at most k or outputs a feedback vertex set of size at most $3k$.*

In fact, TFVS has a polynomial time factor 2.5-approximation, due to Cai et al. [CDZ00]. However, the simpler algorithm from Lemma 5.2 is already suitable to our needs.

The preliminary phase of our algorithm for TFVS is the kernel of Dom et al. [DGH$^+$10]. We will need some additional properties of this kernel that we state here. Essentially, Lemma 5.3 allows us to focus on the case when the number of vertices in the input tournament is $O(k^3)$.

Lemma 5.3. *[DGH$^+$10] There is a polynomial time algorithm that given as input a tournament T and integer k, runs in polynomial time and outputs a tournament T' and integer k' such that $|V(T')| \leq |V(T)|$, $|V(T')| = O(k^3)$, $k' \leq k$, and T' has*

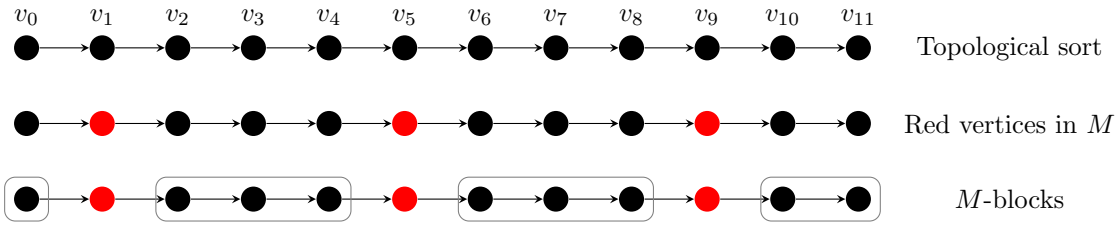

Figure 5.1: Topological sort of $T = v_0 \dots v_{11}$, $M = \{v_1, v_5, v_9\}$ and M-blocks are $\{\{u_0\}, \{u_2, u_3, u_4\}, \{u_6, u_7, u_8\}, \{u_{10}, u_{11}\}\}$

a feedback vertex set of size at most k' if and only if T has a feedback vertex set of size at most k.

Definition 5.4. Let D be a directed graph. For any pair of vertices $u, v \in V(D)$ the set *between*(D, u, v) is defined as $N^+(u) \cap N^-(v) \setminus \{u, v\}$.

Observe that for an acyclic tournament T, *between*(T, u, v) is exactly the set of vertices coming after u and before v in the unique topological ordering of T.

Definition 5.5. Let D be a directed graph and $M \subseteq V(D)$. Two vertices $u, v \in M$ are called *M-consecutive* if $uv \in E(D)$ and *between*$(D, u, v) \cap M = \emptyset$.

In an acyclic tournament T and vertex set M, two vertices u and v in M are M-consecutive if no other vertex of M appears between u and v in the topological ordering.

Definition 5.6. Let D be a directed graph and $M \subseteq V(D)$. We define the set of M-blocks in D. Each pair of *M-consecutive* vertices u and v defines the M-block *between*(D, u, v). Further, each vertex $u \in M$ with no in-neighbors in M defines an M-block $N^-(u)$. Each vertex $u \in D$ with no out-neighbors in M defines the M-block $N^+(u)$. The *size* of an M-block is its cardinality.

In an acyclic tournament T the M-blocks form a partition of $V - M$, where two vertices are in the same block if and only if no vertex of M appears between them in the topological order of T. For example, consider an acyclic tournament $T = u_0u_1...u_{11}$ where vertices are topologically sorted. Let $M = \{u_i | i \mod 4 = 1\}$. *between*$(T, u_1, u_9) = \{u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$. u_5 and u_9 are M-consecutive and $\{u_6, u_7, u_8\}$ is an M-block. The set of all M-blocks in T is $\{\{u_0\}, \{u_2, u_3, u_4\}, \{u_6, u_7, u_8\}, \{u_{10}, u_{11}\}\}$. See Figure 5.1.

5.3 Scattered Pivots

Any induced subgraph M of an acyclic tournament T is an acyclic tournament. As the topological sort of an acyclic tournament is unique, it is useful to observe that the topological sort of M can be obtained by deleting the vertices belonging to $T - M$ from the topological sort of T. Conversely, the topological sort of T can be obtained by *extending* the topological sort of M.

Given a tournament T and an integer k, our goal is to check if there exists a *small* set H such that $T - H$ is acyclic. By the above discussion, if a set M is known with the promise that a feedback vertex set H for T exists which is disjoint from M, we can correctly *extend* the topological sort of M to the topological sort of *most* of the vertices in T. For example, let M be the set of vertices of $T - H$ whose position in the topological order is congruent to 0 mod $\log^2 k$. Such a set would be disjoint from H such that, in the topological order of $T - H$ the distance between two consecutive vertices of M is $O(\log^2 k)$. Of course there is a catch; we defined M using the solution H, but we want to use M to find the solution H. The next lemma shows how to find a set M with the above properties without knowing the optimum feedback vertex set H in advance.

Lemma 5.7. *Let T be a tournament with $|V(T)| = O(k^3)$ where k is an integer. Then, there exists a family $\mathcal{M}$ of subsets of $V(T)$ with $|\mathcal{M}| = 2^{O(\frac{k}{\log k})}$ such that for every feedback vertex set H of T of size at most k, there is a set $M \in \mathcal{M}$, such that :*

1. *$M \cap H = \emptyset$, and*
2. *the size of any M-block in $T - H$ is at most $2\log^2 k$.*

Furthermore, there exists an algorithm that enumerates $\mathcal{M}$ in $2^{O(\frac{k}{\log k})}$ time and polynomial space.

Proof. Let X be a feedback vertex set of T of size at most $3k$ obtained using Lemma 5.2. Let $Y := V(T) \setminus X$ and $v_0 v_1 ... v_{|Y|-1}$ be the topological sort of $T[Y]$ such that the edges in $T[Y]$ are directed from left to right. Partition Y using $\lfloor \log^2 k \rfloor$ colors such that for each $i \in [0, ..., |Y| - 1]$, v_i gets color i mod $\lfloor \log^2 k \rfloor$. For each $c \in [0, ..., \lfloor \log^2 k \rfloor - 1]$, let Y_c be the set of vertices in Y which get color c. Each $M \in \mathcal{M}$ is specified by a 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ where

- c is a color in the above coloring of Y,
- $\hat{H} \subseteq Y_c$ such that $|\hat{H}| \leq \frac{k}{\log^2 k}$,
- $\hat{R} \subseteq Y \setminus Y_c$, $|\hat{R}| \leq |\hat{H}|$, and
- $\hat{X} \subseteq X$ such that $|\hat{X}| \leq \frac{3k}{\log^2 k}$.

For each 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$, let $M := (Y_c \setminus \hat{H}) \cup \hat{R} \cup \hat{X}$. Hence, $|\mathcal{M}|$ is upper bounded by the maximum number of such 4-tuples which is c times the product of total number of choices sets of $\hat{H}, \hat{R}$ and $\hat{X}$.

$$\begin{aligned} |\mathcal{M}| &\leq 2^{\log(\log^2 k)} \times k^{\frac{6k}{\log^2 k}} \times (3k)^{\frac{3k}{\log^2 k}} \\ &= 2^{O(\frac{k}{\log k})} \end{aligned}$$

Clearly, all such 4-tuples can be enumerated in polynomial space thereby providing an enumeration of $\mathcal{M}$.

We prove the correctness of the above algorithm by showing that for every feedback vertex set H of T of size at most k, $\mathcal{M}$ contains a set M which satisfies the properties listed in the statement of the lemma. Let H be an arbitrary feedback vertex set of T of size at most k. Corresponding to this set H, we show that there exists an appropriate 4-tuple.

For each color $j \in [0, ..., \lfloor \log^2 k \rfloor - 1]$, let $H_j := Y_j \cap H$. As $|H| \leq k$, by averaging, there is a color c such that $0 < |H_c| \leq \frac{k}{\log^2 k}$. For this color c, let $\hat{H} := H_c$. Consider a set $\hat{R}$ obtained as follows: for every vertex $v \in H_c$, pick the first vertex *after* v (if there is any) in $Y \setminus (Y_c \cup H)$ in the topological ordering of $T[Y]$. Note that $T[X \setminus H]$ is acyclic. Color $X \setminus H$ using $\lfloor \log^2 k \rfloor$ colors as was done for Y. Let $\hat{X}$ be the set of all vertices colored 0 in this coloring. The size of any $\hat{X}$-block in $T[X \setminus H]$ is $\lfloor \log^2 k \rfloor$. Clearly, $|\hat{X}| \leq \frac{3k}{\log^2 k}$.

The 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ described above satisfies all the properties listed in the construction of $\mathcal{M}$. Let $M := (Y_c \setminus H_c) \cup \hat{R} \cup \hat{X}$. Clearly, $M \cap H = \emptyset$ and $M \in \mathcal{M}$. Since the size of any $[(Y_c \setminus H_c) \bigcup \hat{R}]$-*block* in Y is at most $\log^2 k$, the size of any M-block in $T - H$ is at most $2 \log^2 k$ which concludes the proof of the lemma. □

Set $M \in \mathcal{M}$ acts as a set of *pivots around* which we *arrange* rest of the vertices in $T - M$. But, there are a few shortcomings of Lemma 6.25. The size bound on M-blocks in $T - H$ depends on H and as H is unknown it does not help immediately. Even though with Lemma 6.25, for any feedback vertex set H of size at most k we will find a set M such that the M-blocks in $T - H$ are small, the M-blocks of T do not have to be small, because they could contain many vertices from H. This works in our favor as from *large* blocks we can easily guess the vertices that do not belong to H and branch on their choices. The next lemma provides us with an additional set P corresponding to M that helps us work without the knowledge of H.

Definition 5.8. Let D be a directed graph. A vertex $v \in V(D)$ is *consistent* with a set $M \subseteq V(D)$ if there are no cycles in $D[M \cup v]$ containing v.

Define a function $\mathcal{I}$ that given a directed graph D and a set $M \subseteq V(D)$ outputs a set of vertices *inconsistent* with M. Define another function $\mathcal{L}$ that given a directed graph D, a set $M \subseteq V(D)$ and an integer k outputs a set of vertices which is the union of all *large* M-blocks i.e. of size at least $2 \log^4 k$ in $D - \mathcal{I}(D, M)$.

Lemma 5.9. *Let T be a tournament with $|V(T)| = O(k^3)$ where k is an integer. Then, there exists a family $\mathcal{X} = \{(M_1, P_1), (M_2, P_2, \ldots, (M_l, P_l))\}$ of set pairs of $V(T)$ with $|\mathcal{X}| = 2^{O(\frac{k}{\log k})}$ such that for every feedback vertex set H of T of size at most k, there is a set $(M, P) \in \mathcal{M}$, such that :*

1. *$M \cap H = \emptyset$,*
2. *$P \subseteq H$,*
3. *every vertex of $V(T) \setminus P$ is consistent with M,*

4. *the size of any M-block in $T - H$ is at most $2\log^2 k$, and*

5. *the size of any M-block in $T - P$ is at most $2\log^4 k$.*

Furthermore, there exists an algorithm that enumerates $\mathcal{X}$ in $2^{O(\frac{k}{\log k})}$ time and polynomial space.

Proof. Use the algorithm of Lemma 5.7 to compute $\mathcal{M}$. For each $M \in \mathcal{M}$ compute the sets $\mathcal{I}(T, M)$ and $\mathcal{L}(T, M, k)$. For each $B \subseteq \mathcal{L}(T, M, k)$ such that $|B| \leq \frac{2k}{\log^2 k}$ output a pair of sets $(M, P) = (M, \mathcal{I}(T, M) \cup \mathcal{L}(T, M, k) \setminus B)$. The set $\mathcal{X}$ is the collection of all such pair of sets.

We prove that the algorithm satisfies the stated properties. Consider a feedback vertex set H of size at most k. By Lemma 5.7 there exists $M \in \mathcal{M}$ such that $M \cap H = \emptyset$. Let $C = \mathcal{I}(T, M)$ be the set of vertices that are not consistent with M. These vertices must belong to H. Since, for every vertex $v \in T - C$, $T[M \cup v]$ is an acyclic tournament, v can be placed uniquely in the topological ordering of $T[M]$. Hence, for each $v \in T - C$, there is an unique M-block containing it. Since the size of any M-block in $T - H$ is at most $2\log^2 k$, the size of each M-block in $T - C$ will be at most $k + 2\log^2 k$. An M-block is called *large* if its size is at least $2\log^4 k$. From each *large* M-block at least $2\log^4 k - 2\log^2 k$ vertices belong to H. Hence, in total at most $\frac{k}{2\log^4 k - 2\log^2 k} \times 2\log^2 k \leq \frac{2k}{\log^2 k}$ vertices from the union of *large* M-blocks do not belong to H. Since the algorithm loops over all choices of subsets $B \subseteq \mathcal{L}(T, M, k)$, $|B| \leq \frac{2k}{\log^2 k}$, $\mathcal{X}$ contains a pair (M, P) satisfying the properties listed in the lemma. Moreover, $|\mathcal{X}|$ is bounded by the product of $|\mathcal{M}|$ and the number of subsets B. Now $|\mathcal{L}(T, M, k)| \leq |V(T)|$ which implies the number of subsets B is at most $(k^3)^{\frac{2k}{\log^2 k}} = 2^{3\log k \times \frac{2k}{\log^2 k}} = 2^{O(\frac{k}{\log k})}$. Hence, $|\mathcal{X}| \leq 2^{O(\frac{k}{\log k})} \times 2^{O(\frac{k}{\log k})} = 2^{O(\frac{k}{\log k})}$ □

Observe that the algorithm of Lemma 5.9 does not store the family $\mathcal{X}$, but enumerates all the pairs $(M, P) \in \mathcal{X}$. Furthermore, we can work with the M-blocks in $T - P$. The algorithm for TFVS will go through all pairs in $(M, P) \in \mathcal{X}$ and for each such pair (M, P) search for a feedback vertex set H of size at most k such that (M, P) satisfy the conclusion of Lemma 5.9 for H. In the next section we shall see that the extra restrictions imposed on H by M and P make it easier to find H.

5.4 Main Algorithm for TFVS

In this section we consider the following problem. We are given as input a tournament T and an integer k, and a pair (M, P) of vertex set in T. The objective is to find a feedback vertex set H of T of size at most k, such that (M, P) satisfy the conclusion of Lemma 5.9.

The pair (M, P) naturally leads to a partition of the vertices of $T - (P \cup M)$ into *local subtournaments* corresponding to the induced graphs on the M-blocks in $T - P$. At this point the triangles in $T - P$ can be classified into two types:

those that are entirely within a subtournament and those whose vertices are *shared* between more than one subtournament. The algorithm eliminates all the shared triangles first. When there are no such triangles left, it is possible to solve the problem independently on each of the subtournaments. Since the subtournaments are small, even brute force search is fast enough.

To formalize our approach it is convenient to define an intermediate problem, and interpret the search for a feedback vertex set H such that (M, P) satisfies the conclusion of Lemma 6.39 as an instance of this intermediate problem. Let d and t be two positive integers. Consider a class of mixed multigraphs $\mathcal{G}(d,t)$ in which each member is a mixed multigraph $\mathcal{T}$ with the vertex set $V(\mathcal{T})$ partitioned into vertex sets $V_1, V_2, ..., V_t$ such that for each $i \in [t]$, $|V_i| \leq d$ and $T_i := \mathcal{T}[V_i]$ is a *supertournament* and the undirected edge set is $\mathcal{E}(\mathcal{T}) \subseteq \bigcup_{i<j} V_i \times V_j$.

d-Feedback Vertex Cover (d-FVC)

Input:	A mixed multigraph $\mathcal{T} \in \mathcal{G}(d,t)$, positive integer k
Question:	Does there exists a set $H \subseteq V(\mathcal{T})$ such that $\lvert H\rvert \leq k$ and $\mathcal{T} - H$ is acyclic and contains no undirected edges?

Now we show how TFVS reduces to solving d-FVC.

Lemma 5.10. *There exists a polynomial time algorithm that given a TFVS instance (T,k) and a subset $M \subseteq V(T)$ outputs a d-FVC instance $(\mathcal{T},k)$ such that T has a feedback vertex set H disjoint from M, $P \subseteq M$ and $|S| \leq k$ if and only if $(\mathcal{T},k)$ is a* YES*-instance of d-FVC.*

Proof. We describe an algorithm that reduces T to $\mathcal{T}$ on the same set of vertices as in $T - M$. If $T[M]$ is not acyclic or there is a vertex in T inconsistent with M, then output a trivial NO-instance. Otherwise, let $\mathcal{B} := \{B_1, B_2, ..., B_t\}$ be the set of M-blocks in T such that the elements in $\mathcal{B}$ are indexed according to the topological order of $T[M]$ in which the edges in $T[M]$ are directed from left to right. Let $V(\mathcal{T}) := V(T) \setminus M$. The directed edge set $E(\mathcal{T})$ is $E(T) \setminus \{e \mid \forall i,j \in [t], i \neq j$ and $e \in B_j \times B_i\}$. The undirected edge set in $\mathcal{T}$ is $\mathcal{E}(\mathcal{T}) := \{\text{undirected}(e) \mid i,j \in [t], i < j$ and $e \in B_j \times B_i\}$ where undirected(e) is an undirected edge between the endpoints of e.

Now we argue about the correctness. Since $\mathcal{T}$ is essentially a subgraph of $T - M$ with some additional undirected edges, we use the same symbol to refer to vertex or directed edge sets in both the instances. Suppose S is a feedback vertex cover of $\mathcal{T}$. Clearly S is disjoint from M. We claim that S is a feedback vertex set of T. The triangles in $T - M$ are of two types: ones whose endpoints lie entirely in B_i for some i and others whose endpoints are shared among multiple M-blocks. Clearly, S hits all the triangles within each subtournament $T[B_i]$ in $\mathcal{T}$. Hence, all that remains to show is that S is also a hitting set for all triangles between different subtournaments $T[B_i]$. For the sake of contradiction suppose that there is a triangle $uvwu$ in $T - M - S$ such that not all of u, v and w belong to the same subtournament of $\mathcal{T}$. Then at least one edge ab in this triangle is

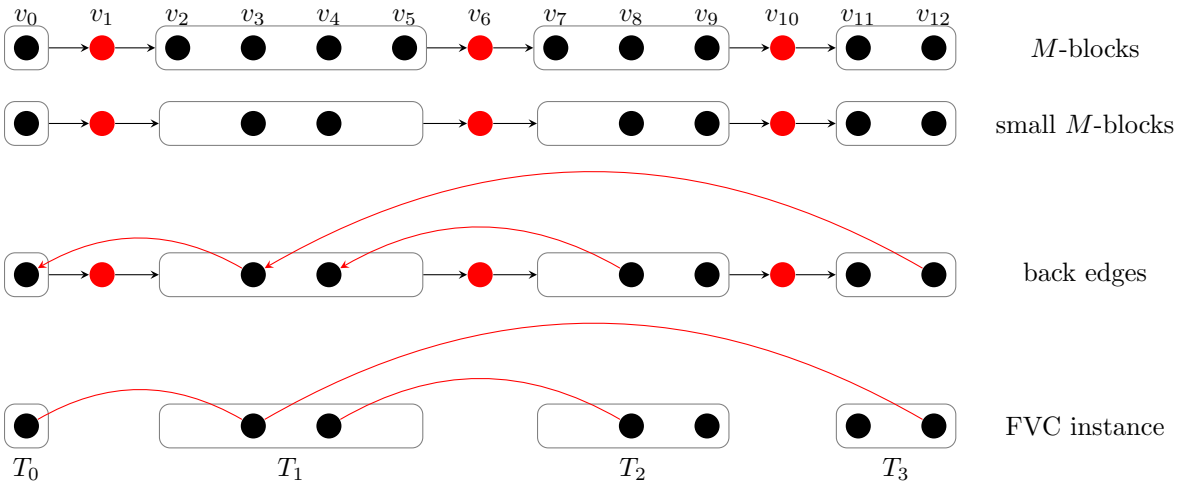

Figure 5.2: From TFVS to FVC

such that $a \in B_i, b \in B_j$ and $i > j$. But by the construction of $\mathcal{E}(\mathcal{T})$ there is an undirected edge between a and b implying that at least one of a or b belongs to S, a contradiction.

In the other direction, suppose S is a feedback vertex set of T disjoint from M. Clearly, S hits all the triangles within each subtournament $T[B_i]$ in $\mathcal{T}$. Hence, all that remains to show is that S is a hitting set for $\mathcal{E}(\mathcal{T})$. Suppose not. Then there is an undirected edge $e = uv \in \mathcal{E}$ which is not hit by S. Consider the directed edge in T corresponding to e. Without loss of generality, we can assume that $u \in B_i$ and $v \in B_j$ for some $i, j \in [t]$ such that the directed edge is from u to v and $i > j$. Now in T, there is a vertex $w \in M$ which lies *after* all elements of B_j and *before* all vertices of B_i and forms a triangle $vwuv$. Since, $w \notin S$, either $u \in S$ or $v \in S$, a contradiction. □

Figure 5.2 shows an example of the reduction from a TFVS instance to one of many FVC instances. In light of Lemma 5.10 we need an efficient algorithm for d-FVC. Next we will give an efficient algorithm for d-FVC and show how it can be used to obtain our claimed algorithm for TFVS. Our algorithm for FVC is based on branching on vertices that appear in at least two edges of $\mathcal{E}(\mathcal{T})$. The case when there are no such vertices has to be handled separately, the algorithm for this case is deferred to Section 5.6. For now, we simply state the existence of the algorithm for this case, and complete the argument using this algorithm as a black box.

Lemma 5.11. *There exists an algorithm running in* $1.5874^s \cdot 2^{O(d^2 + d \log s)} \cdot n^{O(1)}$ *time which finds an optimal feedback vertex cover in a mixed multigraph* $\mathcal{T} \in \mathcal{G}(d, t)$ *in which the undirected edge set* $\mathcal{E}(\mathcal{T})$ *is disjoint and* $|\mathcal{E}(\mathcal{T})| = s$.

The proof of Lemma 5.11 can be found in Section 5.6. Armed with Lemma 5.11 we can give a simple and efficient algorithm for d-FVC. The algorithm is based on branching. In the course of the branching we will sometimes conclude (or guess) that a vertex v is *not* put into the solution H. The operation described below encapsulates the effects of making a vertex undeletable.

In a mixed multigraph D, for any vertex v, D/v is a mixed multigraph obtained by adding a directed edge uw in $D - v$ for every $u \in N^-(v)$ and $w \in N^+(v)$. The next lemma shows that looking for a solution disjoint from v amounts to putting all the undirected neighbors $N_{\mathcal{E}}(v)$ of v into the solution, and finding the optimum solution of $(\mathcal{T} - N_{\mathcal{E}}(v))/v$.

Lemma 5.12. *Let $(\mathcal{T}, k)$ be a d-FVC instance. If for any vertex $v \in V(\mathcal{T})$ such that $N_{\mathcal{E}}(v) = \emptyset$, then $(\mathcal{T}, k)$ has a solution of size at most k not containing v if and only if $(\mathcal{T}/v, k)$ is a* YES*-instance.*

Proof. Let S be a feedback vertex cover of $\mathcal{T}$ of size at most k not containing v. We show that S is a feedback vertex cover of $\mathcal{T}/v$. Clearly, S hits every undirected edge in $\mathcal{T}/v$. For the sake of contradiction suppose there is a cycle of length 3 containing v in $\mathcal{T}/v$ not hit by S. If this cycle is in $\mathcal{T} - v$, then it is hit by S. Hence, the triangle must contain an edge not in $\mathcal{T} - v$. Note that $\mathcal{T}/v$ is obtained by adding a directed edge yx for every triangle $xyvx$ in $\mathcal{T} - v$ thereby creating a 2-cycle between x and y. Since $v \notin S$, either $x \in S$ or $y \in S$, a contradiction. For the same reason there are no cycles of length 2 in $\mathcal{T}/v - S$.

Now suppose S is a feedback vertex cover of $\mathcal{T}/v$. Since every cycle in $\mathcal{T} - v$ is a cycle in $\mathcal{T}/v$ which are hit by S, we need to consider cycles in $\mathcal{T}$ containing v. But, for every such cycle $xyvx$ we have a cycle xyx of length 2 in $\mathcal{T}/v$ which is hit by S, we have that $\mathcal{T} - S$ is acyclic. □

Lemma 5.13. *There exists an algorithm for d-FVC running in $1.466^n \cdot 2^{O(d^2 + d \log n)}$ time and in $1.618^k \cdot 2^{O(d^2 + d \log k)} \cdot n^{O(1)}$ time.*

Proof. We describe a recursive algorithm which searches for a potential solution S of size at most k by branching. For any vertex v, let $N_{\mathcal{E}}(v)$ denote the set of vertices w such that $vw \in \mathcal{E}$. Let $s = |\mathcal{E}|$. As long as there is a vertex v such that $|N_{\mathcal{E}}(v)| \geq 2$ and $k > 0$, the algorithm branches by considering both the possibilities: either $v \in S$ or $v \notin S$. In the branch in which v is picked, n and k are decreased by 1 each and v is removed from the graph. In the other branch, $N_{\mathcal{E}}(v)$ is added to S, and k is decreased by $|N_{\mathcal{E}}(v)|$. At the same time, $N_{\mathcal{E}}(v)$ is removed from the graph. Since, $N_{\mathcal{E}}(v) = \emptyset$, by Lemma 5.12 $(\mathcal{T}, k)$ is reduced to $(\mathcal{T}/v, k)$. Thus the number of vertices is decreased by $|N_{\mathcal{E}}[v]|$. The algorithm stops branching further in a branch in which either $k < 0$ or $k > 0$ and for every vertex v, $|N_{\mathcal{E}}(v)| \leq 1$. In the case that $k < 0$, the algorithm terminates the branch and moves on to other branches. In the other case, if $|\mathcal{E}(T)| > k$, the algorithm terminates that branch, otherwise the algorithm of Lemma 5.11 is applied. If the size of the optimal solution of Lemma 5.11 is at most k, then the algorithm outputs YES and terminates, otherwise the algorithm moves to another branch. If the algorithm fails to find any solution of size at most k in every branch, it outputs NO.

Now we do the running time analysis of the algorithm. At each internal node of the recursion tree, the algorithm spends polynomial time. At a leaf node, either the algorithm terminates or makes a call to the algorithm of Lemma 5.11 with

parameter $s = |\mathcal{E}(\mathcal{T})|$ which is at most k. So, we need to bound the number of times Lemma 5.11 is called with parameter s for each value of s in $[k]$. Note that for any s, the smallest value of k with which a call to the algorithm of Lemma 5.11 is made is s. Therefore, for each value of $s \in [k]$, the number of calls to the algorithm of Lemma 5.11 is bounded by the number of nodes in the recursion tree with $k = s$. The recurrence relation for bounding the number of leaves in the recursion tree of the algorithm is given by:

$$f_s(k) \leq f_s(k-1) + f_s(k-2)$$

which solves to $f_s(k) \leq 1.618^{k-s}$ as $f_s(k) \leq 1$ for $k = s$. Hence, the running time of the algorithm is upper bounded by $\sum_{s=1}^{k} 1.618^{k-s} \times 1.5874^s \cdot 2^{O(d^2+d\log s)} \cdot n^{O(1)} \leq 1.618^k \cdot 2^{O(d^2+d\log k)} \cdot n^{O(1)}$.

We can do a similar analysis to bound the running time in terms of n. Note that in direct correspondence with the fact that when ever k decreases by 1, n decreases by 1 and whenever k decreases by $x \geq 2$, n decreases by $x+1$, we get the following recurrence relation:

$$f_s(n) \leq f_s(n-1) + f_s(n-3)$$

implying $f_s(n) \leq 1.466^{n-s}$ as $f_s(k) \leq 1$ for $n = s$. If s is the size of of the graph, then the largest value of $|\mathcal{E}|$ with which a call to the algorithm of Lemma 5.11 is made, is at most $\frac{s}{2}$. Hence, the running time of the algorithm is upper bounded by $\sum_{s=1}^{n} 1.466^{n-s} \times 1.5874^{\frac{s}{2}} \cdot 2^{O(d^2+d\log n)} \cdot n^{O(1)} \leq 1.466^n \cdot 2^{O(d^2+d\log n)} \cdot n^{O(1)}$. □

Having shown an efficient algorithm for d-FVC, we are now in position to prove our main theorem.

Theorem 15. *There exists an algorithm for TFVS running in $O(1.466^n)$ time and in $O(1.618^k + n^{O(1)})$ time.*

Proof. The algorithm begins by running the kernelization algorithm of Lemma 5.3 for the given TFVS instance. In the remainder we assume that $n = O(k^3)$. Next the algorithm proceeds to apply Lemma 5.9 to create a family of set pairs $\mathcal{X}$. For each set pair $(M, P) \in \mathcal{X}$ it determines whether there is a feedback vertex set H of size at most k such that $H \cap M = \emptyset$ and $P \subseteq H$ as follows:

First it runs the algorithm of Lemma 5.10 with input $(T - P, k - |P|)$ and M to reduce the problem to an equivalent d-FVC instance which is then passed to the algorithm of Lemma 5.13 as input. The algorithm outputs YES and terminates if the output of the algorithm of Lemma **??** is YES. If no solution of size at most k is obtained for any set pair in $\mathcal{X}$, the algorithm outputs NO and terminates.

The correctness of the algorithm follows from Lemma 5.9 and Lemma 5.10. The running time of the algorithm is upper bounded by $|\mathcal{X}|$ times (the running time of the algorithm of Lemma 5.13). Since $|\mathcal{X}| = 2^{o(k)}$ and the bulk of the algorithm is run on a tournament with at most $O(k^3)$ vertices the total time used by the algorithm is upper bounded by $O(1.466^n)$ and $O(1.618^k + n^{O(1)})$. □

We have now proved our main result, assuming the correctness of Lemma 5.11. The remainder of the chapter is devoted to proving Lemma 5.11. The engine of the algorithm of Lemma 5.11 is a new graph partitioning theorem. The next section contains the statement and proof of this theorem, while Section 5.6 wraps up the proof of Lemma 5.11, thereby completing the proof of Theorem 15.

5.5 Balanced Edge Partition Theorem

Given an undirected graph $G, |E(G)| = m$, if each vertex in $V(G)$ is colored *red* or *blue* uniformly at random, then in expectation there will be $\frac{m}{4}$ *red* edges and $\frac{m}{4}$ *blue* edges, where a red edge is an edge whose both endpoints are red and a blue edge is an edge whose both endpoints are blue. Using Chebysev inequality it can be shown that, with high probability, the number of red or blue edges will be within $O(\sqrt{md})$ of $\frac{m}{4}$ where d is the maximum degree of a vertex in the graph. A proof of this fact is skipped in favor of a local search algorithm which runs in polynomial time and provides a coloring with smaller deviation from the expected value than random coloring.

Theorem 16. *Given an undirected, multigraph without self-loops and isolated vertices G of maximum degree at most d and $|E(G)| = m$, there exists a partition (A, B) of $V(G)$ such that*

- $\frac{m}{4} - \frac{d}{2} \leq |E(G[A])| \leq \frac{m}{4} + \frac{d}{2}$,
- $\frac{m}{4} - \frac{d}{2} \leq |E(G[B])| \leq \frac{m}{4} + \frac{d}{2}$, *and*
- $\frac{m}{2} - d \leq |E(G[A, B])| \leq \frac{m}{2} + d$

where $E(G[A, B])$ is the set of edges with one endpoint in A and other in B. Furthermore, there is a polynomial time algorithm to obtain this partition.

Proof. The following local search algorithm is used to obtain the desired partition: At each step, the algorithm maintains a partition (A, B) of $V(G)$. As long as there exists a vertex $v \in A$ (or $v \in B$) such that moving it to other part decreases the measure $\mu = ||E(G[A])| - \frac{m}{4}| + ||E(G[B])| - \frac{m}{4}|$, the algorithm changes the partition to $(A \setminus v, B \cup v)$ (or $(A \cup v, B \setminus v)$). The algorithm terminates if no vertex can be moved. Since $\mu \leq m$ and in each step, it decreases by at least one, above algorithm terminates in polynomial time.
Correctness: Let $m_A := |E(G[A])|, m_B := |E(G[B])|$, and $m_C := |E(G[A, B])|$. Let $x := m_A - \frac{m}{4}$ and $y := m_B - \frac{m}{4}$ when the algorithm terminates. Then, $\mu = |x| + |y|$. For any vertex v, let a_v denote the number of edges incident on v whose other endpoints are in A and b_v denote the number of edges incident on v whose other endpoints are in B. Clearly, for every vertex v, $a_v + b_v \leq d$. Suppose that a vertex $v \in A$ is moved to B. The new partition is $(A', B') = (A \setminus v, B \cup v)$. Then, $m_{A'} = \frac{m}{4} + x - a_v, m_{B'} = \frac{m}{4} + y + b_v$ and the measure at this partition is $\mu' = |x - a_v| + |y + b_v|$. Define $\delta_A^v := \mu' - \mu = |x - a_v| - |x| + |y + b_v| - |y|$. Similarly,

if a vertex $v \in B$ moves to A creating new partition $(A', B') = (A \cup v, B \setminus v)$, we can define $\delta_B^v := \mu' - \mu = |x + a_v| - |x| + |y - b_v| - |y|$. Note that since the algorithm has terminated, for any vertex $v \in V(G)$, $\delta_A^v \geq 0$ and $\delta_B^v \geq 0$. Then, the claim of the theorem is that $|x| \leq \frac{d}{2}$ and $|y| \leq \frac{d}{2}$. For the sake of contradiction assume the following possible values of x and y:

$x > \frac{d}{2}, y > \frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v = |x - a_v| - |x| + b_v$. Suppose that $x < a_v, \delta_A^v = a_v - x - x + b_v = a_v + b_v - 2x$. But, for every vertex $v \in V$, $a_v + b_v \leq d$ which implies $\delta_A^v < 0$, a contradiction. Hence, for all vertices $v \in A$, $x \geq a_v, \delta_A^v = x - a_v - x + b_v = b_v - a_v$. If $v \in A$ is such that $a_v > b_v$, then $\delta_A^v < 0$, a contradiction. Hence, for all vertices $v \in A, a_v \leq b_v$. Then, $\sum_{v \in A} a_v \leq \sum_{v \in A} b_v \implies 2m_A \leq m_C \implies m_C \geq \frac{m}{2} + 2x > \frac{m}{2} + d$ which is a contradiction.

$x < -\frac{d}{2}, y < -\frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v = |y + b_v| - |y| + a_v$. Suppose that $|y| < b_v, \delta_A^v = b_v - |y| - |y| + a_v = a_v + b_v - 2|y|$. But, for all vertices v, $a_v + b_v \leq d$ which implies that $a_v - 2|y| + b_v < 0$, i.e. $\delta_A^v < 0$, a contradiction. Hence, for every vertex $v \in A$, we have that $|y| \geq b_v$ and therefore, $\delta_A^v = |y| - b_v - |y| + a_v = a_v - b_v$. If $v \in A$ is such that $a_v < b_v$, then $\delta_A^v < 0$, a contradiction. Hence, for all vertices $v \in A, a_v \geq b_v$. This implies that $\sum_{v \in A} a_v \geq \sum_{v \in A} b_v \implies 2m_A \geq m_C \implies m_C \leq \frac{m}{2} + 2x < \frac{m}{2} - d$ which is a contradiction.

$x > \frac{d}{2}, y < -\frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v := \mu' - \mu = |x - a_v| - |x| + |y + b_v| - |y| < 0$ as $|x - a_v| - |x| \leq 0$ and $|y + b_v| - |y| \leq 0$ and at least one of the inequalities is strict, hence a contradiction.

$y > \frac{d}{2}, x < -\frac{d}{2}$: Similar to the previous case.

$x > \frac{d}{2}, |y| \leq \frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Suppose that $x < a_v$, then $\delta_A^v = a_v - 2x + |y + b_v| - |y| \leq a_v - 2x + b_v < 0$, a contradiction. Hence, for every vertex $v \in A$, $x \geq a_v$, then $\delta_A^v = |y + b_v| - |y| - a_v \leq b_v - a_v$. If $v \in A$ is such that $a_v > b_v$, then $\delta_A^v < 0$, a contradiction. Hence, for every vertex $v \in A$, we have that $a_v \leq b_v$. This implies that $\sum_{v \in A} a_v \leq \sum_{v \in A} b_v \implies 2m_A \leq m_C \implies m_C \geq \frac{m}{2} + 2x > \frac{m}{2} + d$ which is a contradiction.

$y > \frac{d}{2}, |x| \leq \frac{d}{2}$: Similar to the previous case.

$x < -\frac{d}{2}, |y| \leq \frac{d}{2}$: Consider moving a vertex $v \in B$ to A. If $|x| \geq a_v$, then $\delta_B^v = a_v - 2|x| + |y - b_v| - |y| \leq a_v - 2|x| + b_v < 0$, a contradiction. So, for every vertex $v \in B$, $|x| < a_v$ and $0 \leq \delta_B^v = |x| - a_v - |x| + |y - b_v| - |y| \leq -a_v + b_v$. Hence, for each vertex $v \in B$, $a_v \leq b_v$. This implies $\sum_{v \in B} a_v \leq \sum_{v \in B} b_v \implies m_C \leq 2m_B \implies m_C \leq \frac{m}{2} + 2y < \frac{m}{2}$ which is a contradiction.

$y < -\frac{d}{2}, |x| \leq \frac{d}{2}$: Similar to the previous case.

Hence, $|x| \leq \frac{d}{2}$ and $|y| \leq \frac{d}{2}$. This implies that $\frac{s}{2} - d \leq m_C \leq \frac{s}{2} + d$. This concludes the proof of the theorem. □

5.6 d-FVC with Undirected Degree at Most One

Now that we are equipped with Theorem 16, we are almost ready to prove Lemma 5.11. First we show a lemma that encapsulates the use of Theorem 16 inside the algorithm of Lemma 5.11.

Lemma 5.14. *There exists a polynomial time algorithm that given a mixed multigraph $\mathcal{T} \in \mathcal{G}(d,t)$ with disjoint undirected edge set $\mathcal{E}(\mathcal{T})$ outputs a partition (X,Y) of $V(\mathcal{T})$ such that there are no directed edge with one endpoint in X and other in Y and*

- $||\mathcal{E}(X) \cap \mathcal{E}| - \frac{s}{4}| \leq \frac{d}{2}$,
- $||\mathcal{E}(Y) \cap \mathcal{E}| - \frac{s}{4}| \leq \frac{d}{2}$ *and*
- $||\mathcal{E}(X,Y) \cap \mathcal{E}| - \frac{s}{2}| \leq d$

where $s = |\mathcal{E}(\mathcal{T})|$ and $\mathcal{E}(X)$ is the set of undirected edges in $\mathcal{T}[X]$, $\mathcal{E}(Y)$ is the set of undirected edges in $\mathcal{T}[Y]$ and $\mathcal{E}(X,Y)$ is the set of undirected edges with one endpoint in X and other in Y.

Proof. Construct an undirected, multigraph Z such that $V(Z) = \{z_i | i \in [t]\}$ and $E(Z) = \{z_i z_j | uv \in \mathcal{E}, u \in V_i, v \in V_j\}$. Run the algorithm of Theorem 16 to get the partition (A,B) of $V(Z)$. Output $X := \bigcup_{i,z_i \in A} V_i$ and $Y := \bigcup_{i,z_i \in B} V_i$. Since $\mathcal{E}$ is disjoint and for each $i \in [t]$, $|V_i| \leq d$, maximum degree of a vertex in Z is at most d. Hence, the correctness of the algorithm and the size bound in the lemma follows from Theorem 16. □

We are now ready to prove Lemma 5.11. For convenience we re-state it here.

Lemma 5.11 *There exists an algorithm running in $1.5874^s \cdot 2^{O(d^2 + d \log s)} \cdot n^{O(1)}$ time which finds an optimal feedback vertex cover in a mixed multigraph $\mathcal{T} \in \mathcal{G}(d,t)$ in which the undirected edge set $\mathcal{E}(\mathcal{T})$ is disjoint and $|\mathcal{E}(\mathcal{T})| = s$.*

Proof. The algorithm maintains a set S which is initialized to the empty set $\emptyset$. If the underlying undirected graph of $\mathcal{T}$ is disconnected, then the algorithm solves each connected component independently and outputs S as the union of sets returned for each component. If $s \leq d$, then S is an optimal solution set obtained by a brute force search in the instance. If $s > d$, the algorithm obtains a partition (X,Y) of $V(\mathcal{T})$ by running the algorithm of Lemma 5.14. Then, it loops over all subsets $C \subseteq \mathcal{E}(X,Y)$, calling itself recursively on $\mathcal{T}\Big[V(\mathcal{T}) \setminus (V_C(X) \cup V_{\mathcal{E}(X,Y) \setminus C)}(Y)\Big]$ and computes $S_C := V_C(X) \cup V_{\mathcal{E}(X,Y) \setminus C)}(Y) \cup S'$ where S' is the set returned at the recursive call. Finally, the algorithm outputs the smallest set S_C over all choices of $C \subseteq \mathcal{E}(X,Y)$.

Now to argue about the correctness of the algorithm, we use induction on $|\mathcal{E}(\mathcal{T})|$. In the base case $|\mathcal{E}(\mathcal{T})| \leq d$, S is an optimal feedback vertex cover. As the induction hypothesis, suppose that the algorithm outputs an optimal solution for $d < |\mathcal{E}(\mathcal{T})| < s$. Consider $|\mathcal{E}(\mathcal{T})| = s$. Note that for any $C \subseteq$

$\mathcal{E}(X,Y)$, S_C is a d-feedback vertex cover as $V_C(X) \cup V_{\mathcal{E}(X,Y)\setminus C)}(Y)$ is a hitting set for $\mathcal{E}(X,Y)$ and by the induction hypothesis, S' is an optimal solution for $\mathcal{T}[V(\mathcal{T}) \setminus (V_C(X) \cup V_{\mathcal{E}(X,Y)\setminus C)}(Y))]$. At the same time, for any $C \subseteq \mathcal{E}(X,Y)$, $|V_C(X) \cup V_{\mathcal{E}(X,Y)\setminus C)}(Y)| = |\mathcal{E}(X,Y)|$ which is the size of the smallest hitting set for $\mathcal{E}(X,Y)$. Let S_o be an optimal solution and $C_o := \mathcal{E}(S_o \cap X, Y \setminus S_o)$. Then, we claim that $|S_{C_o}| = |S_o|$. Clearly, $|S_{C_o}| \geq |S_o|$. Now, $S_o \setminus V_{C_o}(X) \cup V_{\mathcal{E}(X,Y)\setminus C_o)}(Y)$ is a d-feedback vertex cover for $\mathcal{T}[V(\mathcal{T}) \setminus (V_{C_o}(X) \cup V_{\mathcal{E}(X,Y)\setminus C_o)}(Y))]$. Therefore, $|S'| \leq |S_o \setminus (V_{C_o}(X) \cup V_{\mathcal{E}(X,Y)\setminus C_o)}(Y))| = |S_o| - |\mathcal{E}(X,Y)| \implies |S_{C_o}| \leq |S_o|$, thus proving the claim.

Now we proceed to the running time analysis of the algorithm. Let $h(s,d)$ be the maximum number of leaves in the recursion tree of the algorithm when run on an input with parameters s and d. Since, in each recursive call, s decreases by at least 1, the depth of the recursion tree is at most s. In each internal node of the recursion tree, the algorithm spends polynomial time in size of the input and in each leaf, it spends at most $2^{O(d^2)}$ time as the total number of vertices in each connected component of $\mathcal{T}$ is $O(d^2)$. Thus, the running time of the algorithm on any input with parameters s and d is upper bounded by $h(s,d) \times 2^{O(d^2)} \times n^{O(1)}$. To upper bound $h(s,d)$, first note that $h(a,d) + h(b,d) \leq h(a+b,d)$ because $h(a,d)$ and $h(b,d)$ represent the number of leaves of two independent subtrees. Now for each $C \subseteq \mathcal{E}(X,Y)$, in $\mathcal{T}[V(\mathcal{T}) \setminus (V_C(X) \cup V_{\mathcal{E}(X,Y)\setminus C)}(Y))]$, the undirected edge set $\mathcal{E}(X,Y) = \emptyset$. Hence, the algorithm effectively solves $\mathcal{T}[V(\mathcal{T}) \setminus (V_C(X)]$ and $\mathcal{T}[V(\mathcal{T}) \setminus V_{\mathcal{E}(X,Y)\setminus C)}(Y)]$ independently where by Lemma 5.14, the number of undirected edges is at most $\frac{s}{4} + \frac{d}{2}$ for each instance. Again by Lemma 5.14, $|\mathcal{E}(X,Y)| \leq \frac{s}{2} + d$. Hence, the number of choices for $C \subseteq \mathcal{E}(X,Y)$ is at most $2^{\frac{s}{2}+d}$. As we have seen for each C, the algorithm calls itself twice on graphs with the undirected edge set size at most $\frac{s}{4} + \frac{d}{2}$. So in total, the algorithm makes $2^{\frac{s}{2}+d+1}$ recursive calls with *parameter* $\frac{s}{4} + \frac{d}{2}$. Thus $h(s,d)$ is upper bounded by the recurrence relation $h(s,d) \leq 2^{1+\frac{s}{2}+d} h(\frac{s}{4} + \frac{d}{2}, d)$ which solves to $h(s,d) = 1.5874^s \cdot 2^{O(d \log s)}$. Hence, the running time of the algorithm is bounded by $1.5874^s \cdot 2^{O(d \log s)} \times 2^{O(d^2)} \times n^{O(1)} = 1.5874^s \cdot 2^{O(d^2 + d \log s)} \cdot n^{O(1)}$. □

The proof of Lemma 5.11 completes the proof of our main result, an algorithm for TFVS with running time upper bounded by $O(1.466^n)$ and by $O(1.618^k + n^{O(1)})$.

Proposition 5.15. *[FGLS16] If there exists a parameterized algorithm for any vertex deletion problem into a hereditary graph class with running time $c^k n^{O(1)}$, then there exists an exact-exponential-time algorithm for the problem with running time $(2 - \frac{1}{c})^{n+o(n)} n^{O(1)}$.*

The above proposition immediately implies the following theorem.

Theorem 17. *There exists an algorithm for TFVS running in 1.3820^n time.*

Chapter 6

Feedback Vertex Set in Bipartite Tournament

6.1 Introduction

In this chapter, we extend the algorithmic scheme used in the previous chapter to that for bipartite tournaments. A *bipartite tournament* is a directed graph where the vertices are partitioned into two sets A and B, there is an arc connecting every vertex in A with every vertex in B, and there are no edges between vertices of A and vertices of B. Tournaments arise naturally from round-robin competitions whereas bipartite tournaments model a two-team competition in which every player in one team plays against every player of the other team. Here arcs are drawn from the winning to the losing player, and often one seeks to rank the players from "best" to "worst" such that players that appear higher in the ranking beat all lower ranked players they played against. Such an absolute ranking possible only if there are no cycles in the tournament. The size of the smallest feedback vertex set then becomes a measure of how far the tournament is from admitting a consistent ranking. For this reason the structure of cycles and feedback vertex sets in (bipartite) tournaments has been studied both from the perspective of graph theory [BL82, Cla85, FH66] and algorithms.

For bipartite tournaments, finding a feedback vertex set reduces to hitting all cycles of length 4. For this reason the Feedback Vertex Set problem is more computationally tractable on bipartite tournaments than on general directed graphs. Specifically the best known approximation algorithm for Feedback Vertex Set on directed graphs has an approximation factor of $O(\log n \cdot \log \log n)$ [ENSS98], and the problem does *not* admit a constant factor approximation assuming the Unique Games Conjecture [GHM$^+$11]. On bipartite tournaments it is easy to obtain a 4-approximation (see Lemma 6.2). Further, an improved approximation algorithm with ratio 3.5 was obtained by Cai et al. [CDZ00].

Similarly, it was open for a long time whether Feedback Vertex Set on general directed graphs admits an FPT algorithm, that is an algorithm that determines whether there exists a solution of size at most k in time $f(k)n^{O(1)}$. In

2008, Chen et al. [CLL+08] gave an algorithm with running time $O(4^k k^{O(1)} k! nm)$, and it is an outstanding open problem whether there exists an algorithm with running time $2^{O(k)} n^{O(1)}$. For bipartite tournaments, the realization that it is necessary and sufficient to hit all cycles of length 4 yields a simple $4^k n^{O(1)}$ time parameterized algorithm: recursively branch on vertices of a cycle of length 4. Truß [Tru05] gave an improved algorithm with running time $3.12^k n^{O(1)}$, Sasatte [Sas08] further improved the running time to $3^k n^{O(1)}$, while Hsiao [Hsi11] gave an algorithm with running time $2^k n^{O(1)}$. Prior to this work, this was the fastest known parameterized algorithm for Feedback Vertex Set on bipartite tournaments. Our main result is an algorithm with running time $O(1.6181^k + n^{O(1)})$. Using the recent black-box reduction from parameterized to exact exponential time algorithms of Fomin et al. [FGLS16] we also obtain an exponential-time algorithm running in $O(1.3820^n)$ time.

Organization of the chapter. In Section 6.2 we set up definitions and notation, and state a few useful preliminary results. In Section 6.3 we define and prove some properties of M-sequence. In Section 6.4 we define and give an algorithm for Constrained Feedback Vertex Set problem.

6.2 Preliminaries

A **square** in a directed graph is a directed cycle of length 4. A **topological sort** of a directed graph D is a permutation $\pi : V(D) \mapsto [n]$ of the vertices of the graph such that for all edges $uv \in E(D)$, $\pi(u) < \pi(v)$. Such a permutation exists for a directed graph if and only if the directed graph is acyclic. A pair of vertices u, v are called **false twins** if $uv \notin E(D), vu \notin E(D)$ and $N^+(u) = N^+(v), N^-(u) = N^-(v)$. For an acyclic bipartite tournament, the topological sort is unique up to permutation of false twins. If a bipartite tournament is acyclic then it does not contain any squares. It is a well-known and basic fact that the converse is also true, see e.g. [DGH+10].

Lemma 6.1. *[DGH+10] A bipartite tournament is acyclic if and only if it contains no squares.*

Lemma 6.1 immediately gives rise to a folklore greedy 4-approximation algorithm for BTFVS: as long as T contains a square, delete all the vertices in this square.

Lemma 6.2 (folklore)**.** *There is a polynomial time algorithm that given as input a bipartite tournament T and integer k, either correctly concludes that T has no feedback vertex set of size at most k or outputs a feedback vertex set of size at most $4k$.*

In fact, BTFVS has a polynomial time factor 3.5-approximation, due to Cai et al. [CDZ00]. However, the simpler algorithm from Lemma 6.2 is already suitable to our needs. The preliminary phase of our algorithm for BTFVS is the kernel of Dom et al. [DGH+10]. We will need some additional properties of this kernel that

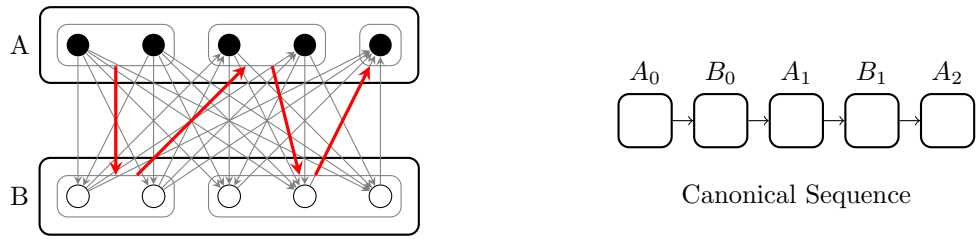

Figure 6.1: Canonical sequence of an acyclic bipartite tournament

we state here. Essentially, Lemma 6.3 allows us to focus on the case when the number of vertices in the input bipartite tournament is $O(k^3)$.

Lemma 6.3. *[DGH+10] There is a polynomial time algorithm that given as input a bipartite tournament T and integer k, runs in polynomial time and outputs a bipartite tournament T' and integer k' such that $|V(T')| \leq |V(T)|$, $|V(T')| = O(k^3)$, $k' \leq k$, and T' has a feedback vertex set of size at most k' if and only if T has a feedback vertex set of size at most k.*

For any sequence σ, let $|\sigma|$ denote the length of σ. For each $i = 1, 2, \ldots, |\sigma|$, let V_i be the i-th element of σ. Let T be an n-vertex acyclic bipartite tournament. The *canonical sequence* for T is the sequence σ of vertex sets that can be obtained from T in $O(n^2)$ time as follows: For each $i \geq 1$, let V_i consist of the vertices without incoming edges in $T \setminus \bigcup_{j=1}^{i-1} V_j$. See Figure 6.1.

Lemma 6.4. *[Hsi11] Let T be an n-node acyclic bipartite tournament. Let σ be the canonical sequence for T. The following statements hold.*

1. *$V_1, V_2, \ldots, V_{|\sigma|}$ form a partition of $V(T)$.*
2. *For each directed edge (u, v) of T, the vertex set V_i containing u precedes the vertex set V_j containing v in the sequence (i.e. $i < j$).*
3. *$A = \bigcup_{i \equiv 1 \mod 2} V_i$ and $B = \bigcup_{i \equiv 0 \mod 2} V_i$ are the partite sets of T.*

Definition 6.5 (t-wise independent)**.** A family $H_{n,t,q}$ of functions from $[n]$ to $[q]$ is called a t-wise independent sample space if, for every t positions $1 < i_1 < i_2 < \cdots < i_t \leq n$, and every tuple $\alpha \in [q]^t$, we have $Pr(f(i_1), f(i_2), \ldots, f(i_t)) = \alpha = q^{-t}$ where the function $f \in H_{n,t,q}$ is chosen uniformly at random.

Theorem 18. *[ABI86] There exists a t-wise independent sample space $H_{n,t,q}$ of size $O(n^t)$ and it can be constructed efficiently in time linear in the output size.*

6.3 M-Sequence

As in the case of tournaments, in this case as well we look for a set M of pivots around which we arrange rest of the vertices. But unlike tournaments, it is not clear how to extend the relative arrangement of vertices in M to vertices in $T - M$

in a meaningful way. For this, we need to relate the canonical sequence of an acyclic bipartite tournament T to the canonical sequence of a subset of vertices $M \subseteq V(T)$. First we extend the notion of the canonical sequence to general bipartite tournaments relative to a set M of vertices.

Definition 6.6 (M-consistent)**.** Let T be a directed graph and $M \subseteq V(T)$. T is called M-consistent if for every vertex $v \in V(T)$ $T[M \cup \{v\}]$ is acyclic.

Definition 6.7 (M-equivalent)**.** Given a directed graph T and a subset $M \subseteq V(T)$, two vertices $u, v \in V(T)$ are said to be M-equivalent if both have same neighborhood in M i.e. if $N^+(u) \cap M = N^+(v) \cap M$ and $N^-(u) \cap M = N^-(v) \cap M$.

Let T be a bipartite tournament and a subset $M \subseteq V(T)$ such that $T[M]$ is acyclic. If $(X_1, X_2, \dots)$ is the canonical sequence of $T[M]$, then by the above definition, for every set X_i, vertices in X_i are M-equivalent. The next definition extends this notion to arbitrary vertices.

Definition 6.8 ((M, X)-equivalent)**.** Let T be a bipartite tournament and a subset $M \subseteq V(T)$ such that $T[M]$ is acyclic. Let $(X_1, X_2, \dots)$ be the canonical sequence of $T[M]$. For any set X_i in the canonical sequence of $T[M]$ and any vertex $v \in V(T)$, v is called (M, X_i)-equivalent if v is M-equivalent to a vertex in X_i.

Note that the above definition allows us to extend the canonical sequence of $T[M]$ to the canonical sequence of $T[M \cup \{v\}]$ i.e. $(X_1, X_2, \dots, X_i, \dots)$ to $(X_1, X_2, \dots, X_i \cup \{v\}, \dots)$. But, it is not always possible to achieve this for every vertex in T. The vertices, for which it is not possible to get such extension, are called conflicting as defined below.

Definition 6.9 ((M, X)-conflicting)**.** Let T be a bipartite tournament and a subset $M \subseteq V(T)$ such that $T[M]$ is acyclic. Let $(X_1, X_2, \dots)$ be the canonical sequence of $T[M]$. For any set X_i in $(X_1, X_2, \dots)$ and for any vertex $v \in V(T)$, v is called (M, X_i)-conflicting if

- $N^+(v) \cap X_i \neq \emptyset$, $N^-(v) \cap X_i \neq \emptyset$, and
- for every $j < i$, $N^+(v) \cap X_j = \emptyset$ and for every $j > i$, $N^-(v) \cap X_j = \emptyset$.

Clearly, the first condition is sufficient to imply that the canonical sequence of $T[M]$ can not be extended to any canonical sequence of $T[M \cup \{v\}]$ in the sense described above. As violation of the second condition implies that v is not M-consistent and since M is *supposed* to be the pivots, v must belong to the solution H we seek. As a direct consequence of the above definitions, we have the following lemma.

Lemma 6.10. *Let T be an M-consistent bipartite tournament for some subset $M \subseteq V(T)$. Let $(X_1, X_2, \dots, X_i, X_{i+1}, \dots)$ be the canonical sequence of $T[M]$. Let $v \in V(T)$ be a (M, X_i)-conflicting vertex. Then, the canonical sequence of $T[M \cup \{v\}]$ is $(X_1, X_2, \dots, X_i', \{v\}, X_i'', X_{i+1}, \dots)$ where $X_i' \cup X_i'' = X_i$ such that $X_i', X_i'' \neq \emptyset$.*

Hence, so far we have that there are two ways we can extend the canonical sequence of $T[M]$ to that of $T[M \cup \{v\}]$: in the first way, v *gets placed* in one of the sets X_i and in the second way, the vertex v *forms* a new set $\{v\}$ in the canonical sequence by *splitting* a set X_i into two nonempty pieces. The next definition is about a special type of vertices, those which will be placed either in the beginning or at the end of the extended canonical sequence.

Definition 6.11 (M-universal)**.** Let T be a bipartite tournament and a subset $M \subseteq V(T)$ such that $T[M]$ is acyclic. Let $(X_1, X_2, \dots)$ be the canonical sequence of $T[M]$. A vertex $v \in V(T)$ is called M-universal if the following holds:

- v is not (M, X_i)-equivalent for any X_i,
- $T[M \cup \{v\}]$ is acyclic.
- There exists a topological sort of $T[M \cup \{v\}]$ such that v is either the first vertex (called M^--universal) or is the last vertex (called M^+-universal) in the ordering.

Now, we have, as the next lemma states, a complete characterization of vertices in T with respect to a set M.

Lemma 6.12. *Let T be an M-consistent bipartite tournament and let $(X_1, X_2, \dots)$ be the canonical sequence of $T[M]$. Then, for every vertex $v \in V(T)$, there exists a unique index i such that v satisfies exactly one of the following properties:*

- *v is (M, X_i)-equivalent,*
- *v is (M, X_i)-conflicting,*
- *v is M-universal.*

Proof. Since T is M-consistent, $T[M \cup \{v\}]$ is acyclic. By definition, v can not satisfy more than one property. If v is M-universal, then, v is neither (M, X_i)-equivalent nor (M, X_i)-conflicting for any set X_i.

If v is (M, X_i)-equivalent to some set X_i, then by definition, v is not M-universal. In addition, for any set X_j, v is not (M, X_j)-conflicting as no vertex in X_i is (M, X_j)-conflicting.

Suppose that v is neither (M, X_i)-equivalent for any X_i nor M-universal. We show that v is (M, X_i)-conflicting the first set X_i that contains an out-neighbor u_i of v. Suppose that there is an index $j > i$ such that X_j contains an in-neighbor u_j of v. Since $j - i \geq 2$, there is an index $i < l < j$ such that X_l lies in the partite set of T different from $X_i \cup X_j$. This gives us a cycle $vu_iu_lu_jv$ where $u_l \in X_l$ contradicting that $T[M \cup \{v\}]$ is acyclic. If every vertex in X_i is an out-neighbor of v, then by definition of the canonical sequence, v is (M, X_{i-1})-equivalent contradicting the above assumption. Hence, X_i contains an in-neighbor of v, thereby proving that v is (M, X_i)-conflicting. □

Now we are ready to generalize the canonical sequence which is defined only for acyclic bipartite tournaments to M-sequence which is defined for any bipartite tournament.

Definition 6.13 (M-sequence)**.** Let T be an M-consistent bipartite tournament and $(X'_1, X'_2, \dots)$ be the canonical sequence of $T[M]$. An M-sequence $(X_1, Y_1, X_2, Y_2, \dots, X_l, Y_l)$ of T is a sequence of subsets $V(T)$ such that for every index i, X_i is the set of all vertices in $V(T)$ that are (M, X'_i)-equivalent and Y_i is the set of vertices that are (M, X'_i)-conflicting. In addition, Y_1 contains every M^--universal vertex and Y_l contains every M^+-universal vertex. For every i, the set $X_i \cup Y_i$ is called a *block*, X_i is called the M-sub-block and Y_i is called the $\bar{M}$-sub-block.

Lemma 6.14. *If T is an M-consistent bipartite tournament, then T has a unique M-sequence.*

Proof. The existence and the uniqueness of M-sequence follows from Lemma 6.12 and the uniqueness of the canonical sequence of $T[M]$. □

As a consequence of Lemma 6.4 and Lemma 6.12, we get the following lemma.

Lemma 6.15. *Let $T := (A, B, E)$ be an M-consistent bipartite tournament and $(X'_1, X'_2, \dots)$ be the canonical sequence of $T[M]$. Let $(X_1, Y_1, X_2, Y_2, \dots)$ be the M-sequence of T. The following statements hold:*

1. *$X_1, Y_1, X_2, Y_2, \dots$ form a partition of $V(T)$*
2. *for each i, $X'_i \subseteq X_i$*
3. *for each i, $Y_i \cap M = \emptyset$*
4. *for every odd i, $X_i \subseteq A, Y_i \subseteq B$ and for every even i, $X_i \subseteq B, Y_i \subseteq A$.*

Definition 6.16 (Refinement)**.** A partition $(V_1, V_2, \dots)$ of U is said to be a refinement of another partition $(V'_1, V'_2, \dots)$ if for every set V_i and V'_j, either $V_i \subseteq V'_j$ or $V_i \cap V'_j = \emptyset$.

Now we show that in case of an acyclic bipartite tournament, for any subset $M \subseteq V(T)$, the canonical sequence is a refinement of the M-sequence.

Lemma 6.17. *Let T be an acyclic bipartite tournament. Then, for any subset $M \subseteq V(T)$, the canonical sequence of T is a refinement of the M-sequence of T.*

Proof. Let $(X'_1, X'_2, \dots)$ be the canonical sequence of $T[M]$ and $(X_1, Y_1, \dots, X_l, Y_l)$ be the M-sequence of T. Let $(V_1, V_2 \dots)$ be the canonical sequence of T. Since each set V_i are V_i-equivalent in T, if any vertex in V_i belongs to X_j, then every vertex in V_i belongs to X_j. If any vertex in V_i is (M, X'_j)-conflicting, then every vertex in V_i is (M, X'_j)-conflicting. Hence, $V_i \subseteq Y_j$. If any vertex in V_i is M-universal, then every vertex in V_i is M-universal. Hence, $V_i \subseteq Y_1$ or $V_i \subseteq Y_l$. The family of sets V_i that contain an M^--universal vertex lie in Y_1 and the family of sets V_i that contain an M^+-universal vertex lie in Y_l. □

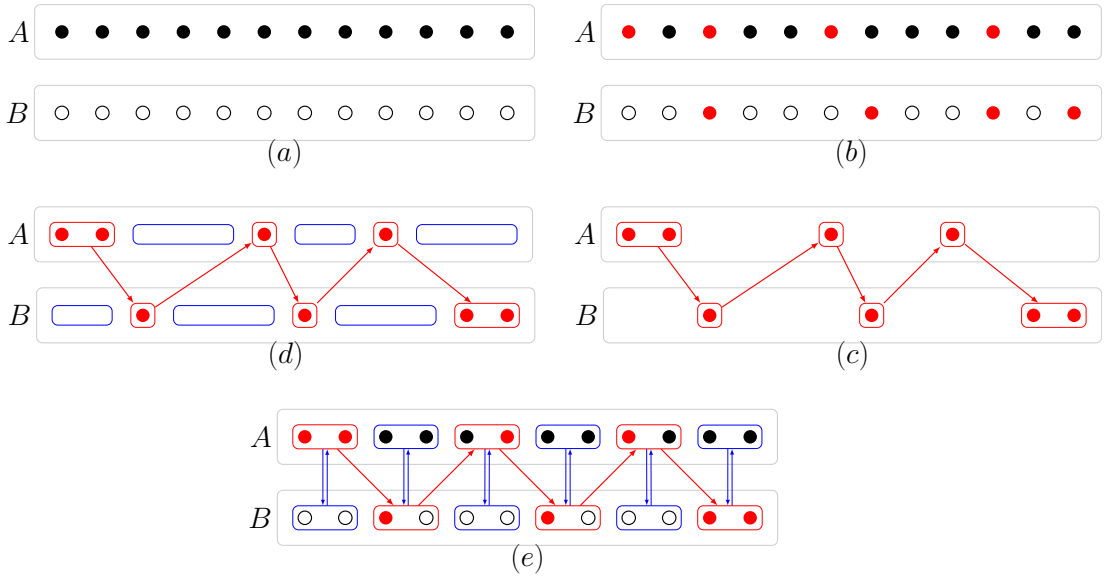

Figure 6.2: (a) A bipartite tournament T, (b) Red vertices belong to M, (c) Canonical sequence of $T[M]$. Red bags correspond to X_i's, (d) M-sequence of $T[M]$ obtained by inserting empty blue bags (correspond to Y_i's) in the canonical sequence of M, (e) Every vertex in T either goes to a red bag or a blue bag (use Lemma 6.18) which provides the M-sequence of T. For clarity, only a few edges have been shown.

The next lemma provides us the tool so that we can construct the M-sequence of T step-by-step from the M-sequence of $T[M]$. See Figure 6.2 for an illustration.

Lemma 6.18. *Let T and $T \cup \{v\}$ be two M-consistent bipartite tournaments and let $(X_1, Y_1, \dots)$ be the M-sequence of T. Then, there exists an index i, such that the M-sequence of $T \cup \{v\}$, is either $(X_1, Y_1, \dots, X_i \cup \{v\}, \dots)$ or $(X_1, Y_1, \dots, Y_i \cup \{v\}, \dots)$.*

Proof. The proof follows from Lemma 6.12. □

Now we can resume our goal to link M-sequence to finding a feedback vertex set in a given bipartite tournament T. Lemma 6.18 implies the following lemma.

Lemma 6.19. *Let T be a bipartite tournament and H be a feedback vertex set of T. Let $M \subseteq T - H$ and $P \subseteq H$. Let $(X_1, Y_2, \dots, X_l, Y_l)$ be the M-sequence of $T - H$ and $(X'_1, Y'_1, \dots, X'_l, Y'_l)$ be the M-sequence of $T - P$. Then, for each index i, $X_i \subseteq X'_i$ and $Y_i \subseteq Y'_i$.*

6.4 Constrained BTFVS

First, we note the essential steps used in the algorithm for feedback vertex set in tournaments described in the previous chapter as our goal is to mimic that algorithm. Throughout the description, we assume that the vertices are arranged

in any topological sort such that all edges go from left to right. In this sense, edges that are directed from right to left in some tentative ordering of vertices are referred to as *back edges*. In the first phase, the algorithm for TFVS partitions the vertex set into small sets.

1. Shrink the input to $O(k^3)$ vertices by using a kernelization algorithm.

2. Uniqueness of the topological sort of an acyclic tournament implies that for any subset M of vertices in $T - H$, the position of other vertices must respect the topological sort of $T[M]$. In fact, the topological sort of $T[M]$ can be seen as the restriction of the topological sort of $T - H$ restricted to the vertices of M. We search for a solution H corresponding to which there is a set M that is *uniformly scattered* over the unique topological sort of $T - H$. Technically, this is captured by having only *small* number of vertices in $T - H$ between any two consecutive vertices of M.

3. Finding a set of vertices $P \subseteq H$ such that M is uniformly scattered over $T - P$ as well. Now M-blocks in $T - P$ contain only bounded number of vertices from H. These blocks are referred to as subtournaments.

Structurally, the first phase can be seen as providing us with a *chain* of subtournaments. There are triangles within each subtournament and between vertices of different subtournaments. In the second phase, the algorithm deals with the triangles of the second type.

1. Uniqueness of the topological sort implies that every edge *going backwards* (from right to left) between vertices of different subtournaments belongs to a triangle with some M-vertex. Hence, the sought solution H must hit these edges. This reduces to a problem called Feedback Vertex Cover.

2. Using branching, the algorithm makes backward edges disjoint.

As the only link between different subtournaments is via these disjoint edges that H must hit, one can simply *guess* one endpoint of each such edge to reduce the graph into a disjoint collection of small subtournaments, for each of which a brute-force search suffices to get the required solution H. But cost of such *guesses* would be $\mathcal{O}(2^k)$. Instead of this, in the final phase, the algorithm uses the partition theorem to branch in a divide and conquer fashion.

In the case of bipartite tournament, we lose the uniqueness of topological sort to some extent and this makes the algorithm very complex. What remains unique for an acyclic bipartite tournament is its canonical sequence. We briefly outline the steps of the algorithm whose formal description will be provided in the rest of the chapter.

Kernel. As before, the first step is to shrink the input to $O(k^4)$ vertices by using a kernelization algorithm.

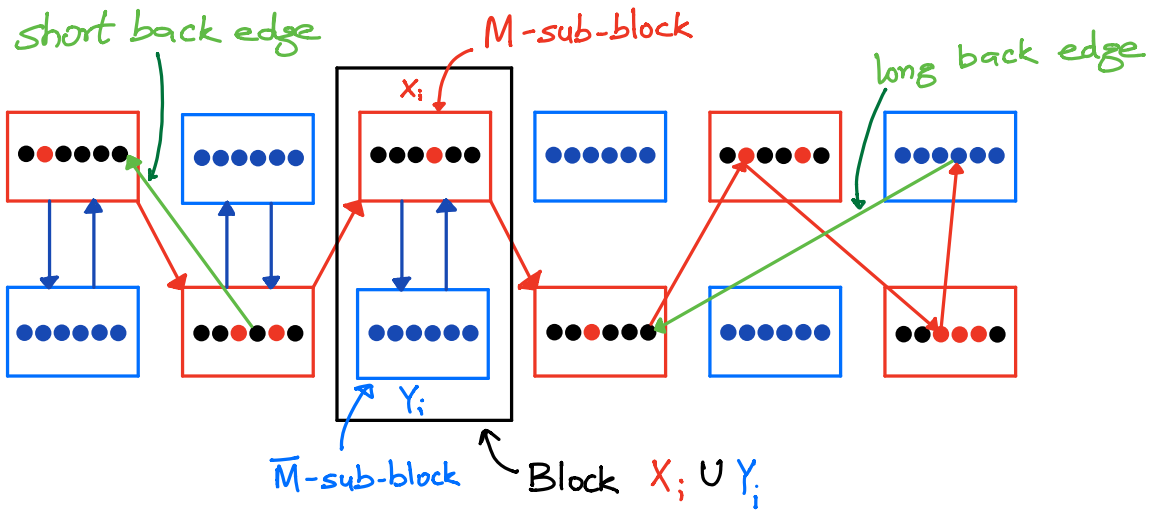

Figure 6.3: M-sequence of T

Scattered pivots. Fix some topological sort π of $T - H$. The set M is chosen such that M is *uniformly scattered* over this topological sort, i.e. between any two consecutive vertices of M, there are only *small* number of vertices from $T - H$. We can not immediately use a topological sort of $T[M]$ to extend to a topological sort of $T - H$ as was possible in the case of tournaments. Using the canonical sequence of $T[M]$, we obtain the M-sequence of T. As a simple intermediate step, vertices forming a square with vertices in M can be immediately deleted i.e. put in H.

Long back edges. We see M-sequence as a *chain* of blocks $X_i \cup Y_i$. There are squares within blocks and between different blocks. But unlike tournaments, edges going backward are not necessarily *conflict* edges, i.e. belong to some square. But we will see that if a backward edge is *too long* (in terms of positions of the endpoints in the M-sequence), it must belong to a square. See Figure 6.3. All *long back edges* are marked as *conflict* edges. But the main difficulty is in recognizing the back edges (called *short back edges*) between *nearby* blocks which do belong to some square.

Weakly-coupled. For short back edges, we deal with adjacent blocks, one pair at a time. Let F be the set of short back edges between blocks $X_i \cup Y_i$ and $X_{i+1} \cup Y_{i+1}$ such that F has a large matching. We observe that most of these edges must be hit by H. Via branching we mark the edges to be hit by H as *conflict* edges. In some sense, this branching makes the adjacent blocks *weakly-coupled*.

Matched. In this step, the algorithm makes all marked conflict edges disjoint via branching over vertices of *conflict degree* at least 2. In this sense, the marked conflict edges form a matching. As we'll see this branching is the bottleneck of the algorithm.

Low block degree. The previous two steps ensure that the matching on short back edges between any pair of blocks is small. But there can still be blocks at which there is a large matching in the set of conflict edges (primarily due to set of long back edges incident on the block). Observing that the vertex cover of short back edges incident on any block is small (due to previous branchings), we can completely guess the set of all conflict edges incident on this block and via a clever branching, we can get rid of such blocks in sub-exponential time. In this sense, blocks now have *low degree* in conflict edges.

Regular. In this step, we branch in each block that has an *excess* number of vertices from H i.e. every block in the M-sequence of $T - H$ has a certain fraction of vertices from M and if this is violated, we branch to fix it.

After this hairy branching phase, we are close to our goal. Now, we have a chain of blocks, that have small number of disjoint conflict edges incident on them although not every conflict edge has been marked (but their number is small). At this point one would be tempted to use the partition theorem. But, the number of blocks can be large. We would be required to completely mark all conflict edges between these blocks. Doing this would be too expensive. Hence to decrease the number of such guesses, we *group* contiguous blocks in *large* bags such that each bag contains a certain minimum number of conflict edges and at the same time has a bounded number of conflict edges incident on the bag from outside the bag. This decreases the number of bags to some polynomial in $\log(k)$. These bags are the equivalent of the super-tournaments in the case of the algorithm for tournament. From this point on the algorithm proceeds as the one for the tournament.

Now we move on to the formal description of the algorithm. As is evident from the outline above, we brach on vertices and edges many times and mark some edges as conflict edges. Hence, it is natural that one should extend the notion of finding feedback vertex set in a bipartite tournament to a constrained version of the problem in which the algorithm is forced to pick certain vertices in the solution and hit certain set of marked edges. With this strategy in mind, we define the Constrained Feedback Vertex Set problem.

Definition 6.20 (Constrained Feedback Vertex Set(CFVS))**.** Let T be a bipartite tournament with vertex subsets $M, P \subseteq V(T)$, edge set $F \subseteq E(T)$. A feedback vertex set H of T is called (M, P, F)-*constrained* if $M \cap H = \emptyset$, $P \subseteq H$ and H is a vertex cover for F.

Constrained Feedback Vertex Set (CFVS)

Input: A bipartite tournament, vertex sets $M, P \subseteq V(T)$, edge set $F \subseteq E(T)$ and positive integer k

Question: Does T has an (M, P, F)-constrained CFVS H of size at most k?

In the rest of the chapter, we assume that the size of the bipartite tournament is at most $O(k^3)$ as a bi-product of the kernelization algorithm (Lemma 6.3). Given a topological sort π of an acyclic bipartite tournament $T = (A, B, E)$, we denote π_A to be the permutation of A when π is restricted to A. Similarly, π_B denotes the permutation of B when π is restricted to B. Next we extend the idea used in Lemma 5.7. Again we want to use a set M that is *uniformly* scattered in some topological sort of $T - H$. To that end, we define a property of a feedback vertex set of a bipartite tournament and while solving for BTFVS, we will look for solutions H that have this property.

Definition 6.21 (M-homogeneous)**.** Let T be a bipartite tournament and k be a positive integer. Let $M \subseteq V(T)$ be a vertex subset such that $T[M]$ is acyclic. A feedback vertex set H of size at most k of T is called M-homogeneous if there exists a topological sort π of $T - H$ such that every subset of $10 \log^3 k$ consecutive vertices in π_{A-H} or π_{B-H} contains a vertex of M.

The algorithm for CFVS is primarily based on branching and often, given a CFVS instance, a family of CFVS instances with addition properties will be constructed. We abstract it out in the following definition.

Definition 6.22 (γ-reduction)**.** A γ-reduction is an algorithm that given a CFVS instance (T, M, P, F, k) outputs in time γ a family $\mathcal{C}$ of size γ of CFVS instances such that

Forward direction if (T, M, P, F, k) has an M-homogeneous (M, P, F)-solution, then there exists an instance $(T, M, P_i, F_i, k) \in \mathcal{C}$ that has an M-homogeneous (M, P_i, F_i) solution.

Backward direction if there exists an instance $(T, M, P_i, F_i, k) \in \mathcal{C}$ that has an (M, P_i, F_i)-CFVS solution, then (T, M, P, F, k) has an (M, P, F)-solution.

Now, we construct a family of sets $\mathcal{M}$ such that if (T, k) has a solution H of size at most k, then there is a set $M \in \mathcal{M}$ such that H is M-homogeneous, and hence we can restrict our attention to looking for feedback vertex sets which are M-homogeneous for some subset M.

Lemma 6.23. *There exists an algorithm that given a bipartite tournament T and a positive integer k outputs in time γ, a family $\mathcal{M}$ of size γ of subsets of $V(T)$ for $\gamma = 2^{O(\frac{k}{\log k})}$ such that for every feedback vertex set H of size at most k of T, there exists $M \in \mathcal{M}$ such that H is M-homogeneous.*

Proof. Using T and k, we construct $\mathcal{M}$. Let $n = |V(T)|, t = 10\log^3 k, q = \log^2 k$. As the first step, the algorithm uses Theorem 18 to construct a family of functions $H_{n,t,q}$ from $[n]$ to $[q]$. Next, the algorithm computes a family $\mathcal{Z}$ of t-wise independent subsets of $V(T)$: For each $f \in H_{n,t,q}$, let $Z := \{v_i \in V(T) \mid f(i) = 1\}$. Add Z to $\mathcal{Z}$. In the next step, for every subset $Z \in \mathcal{Z}$, compute the family of subsets $\mathcal{M}_Z := \{M := Z \setminus \hat{H} \mid \hat{H} \subseteq Z, |\hat{H}| \leq \frac{2k}{\log^2 k}\}$. Finally, output $\mathcal{M} := \bigcup_{Z \in \mathcal{Z}} \mathcal{M}_Z$.

To argue about the correctness of the algorithm, first, we check that the size of $\mathcal{M}$ computed by the above algorithm is consistent with the claim in the lemma. Clearly, $|\mathcal{M}| \leq |H_{n,t,q}| \times |\mathcal{M}_Z| = O(n^t)O((k^3)^{\frac{2k}{\log^2 k}}) = 2^{O(\frac{k}{\log k})}$. We need to show that for every feedback vertex set H of size k and for every topological sort π of $T - H$, there exists a function $f \in H_{n,t,q}$ and a set $\hat{H} \subseteq V(T)$ such that $M := Z \setminus \hat{H}$ satisfies the required properties. Fix a feedback vertex H of size k and a topological sort π of $T - H$. First, we prove the following claim:

Claim 6.24. *If we pick f from $H_{n,t,q}$ uniformly at random, then with non-zero probability, the following two events happen:*

- *for every set of $10\log^3 k$ consecutive vertices in π_{A-H} or π_{B-H}, there is a vertex in Z_f*
- $|Z_f \cap H| \leq \frac{2k}{\log^2 k}$.

Proof. By t-wise independence of $H_{n,t,q}$, the probability that no vertex is picked from t consecutive vertices in π_{A-H} or π_{B-H} is at most $(1 - \frac{1}{q})^t$. Let $\mathcal{A}_1$ be the event that at least one set of t-consecutive vertices either in π_{A-H} or in π_{B-H} does not contain any vertex from Z. Since there at most n sets of t-consecutive vertices, by union bound, the probability that event $\mathcal{A}_1$ happens is at most $n \times (1 - \frac{1}{q})^t \leq Ck^4 \times (1 - \frac{1}{\log^2 k})^{10\log^3 k} = Ck^4 \times \frac{1}{k^{10}} \leq \frac{1}{k^5}$. Let $\mathcal{A}_2$ be the event that at least $\frac{2k}{\log^2 k}$ vertices of H are in Z. The expected number of vertices of H that belong to Z is $k \times \frac{1}{q} = \frac{k}{\log^2 k}$. Therefore, by Markov's inequality, the probability that the event $\mathcal{A}_2$ occurs is at most $\frac{1}{2}$. By union bound the probability that at least one of the events $\mathcal{A}_1$ or $\mathcal{A}_2$ happen is at most $\frac{1}{k^5} + \frac{1}{2}$. Hence, the probability that none of $\mathcal{A}_1$ and $\mathcal{A}_2$ is at least $1 - (\frac{1}{k^5} + \frac{1}{2}) > 0$, thereby implying the claim. □

Hence, the set of functions satisfying the properties in the above claim is non-empty. Let f be such a function. Since, $\mathcal{M}_Z$ is the collection of sets $Z \setminus \hat{H}$ such that $|\hat{H}| \leq \frac{2k}{\log^2 k}$, there exists a choice $\hat{H}$ such that $\hat{H} = Z \cap H$. Hence, $M := Z \setminus \hat{H}$ satisfies the required properties.

For the running time of the algorithm, $H_{n,t,q}$ can be constructed in $O(n^t)$ time. For each function $f \in H_{n,t,q}$, the set Z can be obtained in $O(n)$ time. For each Z, $\mathcal{M}_Z$ can be obtained in $O(k^{\frac{2k}{\log^2 k}})$ time. Hence, the running time of the algorithm is $O(n^t) \cdot n \cdot O(k^{\frac{2k}{\log^2 k}}) = 2^{O(\frac{k}{\log k})}$. □

Once $\mathcal{M}$ has been obtained, for each choice of $M \in \mathcal{M}$, with the assumption that we are looking for an M-homogeneous solution H, we form a CFVS instance

$(T, M, \emptyset, \emptyset)$. In that case, any vertex v that forms a cycle with vertices in M, must be deleted. The following lemma formalizes this.

Lemma 6.25. *There exists an algorithm that given a* BTFVS *instance (T, k) outputs in time γ, a family $\mathcal{C} := \{(T, M_1, P_1, \emptyset, k), (T, M_2, P_2, \emptyset, k), \dots\}$ of size γ of CFVS instances for $\gamma = 2^{O(\frac{k}{\log k})}$ such that*

- *if (T, k) has a feedback vertex set H of size at most k, then $\mathcal{C}$ has a CFVS instance $(T, M, P, \emptyset, k)$ that has an M-homogeneous solution of size at most k and*
- *if $\mathcal{C}$ has a $(M, P, \emptyset)$-constrained solution, then (T, k) has a feedback vertex set of size at most k.*

Proof. Given (T, k), we use the algorithm of Lemma 6.23 with T, k as input and obtain the family of sets $\mathcal{M}$. For each set $M \in \mathcal{M}$, we add a CFVS instance $(T, M, P, \emptyset, k)$ in $\mathcal{C}$ where P is the set of vertices in $V(T) \setminus M$ that form a cycle of length 4 with M. For the forward direction, if (T, k) has a solution H of size at most k, then by Lemma 6.23, there exists a set $M \in \mathcal{M}$ such that H is M-homogeneous. Since, P is the set of vertices that form a cycle with M and $M \cap H = \emptyset$, $P \subseteq H$. Hence, $(T, M, P, \emptyset, k) \in \mathcal{C}$ has an M-homogeneous solution. The backward direction immediately follows from the construction of $\mathcal{C}$. □

Note that M is defined with respect to some set of topological sorts of $T - H$. Next we relate these permutations with the M-sequence. In particular, we want to relate the number of M-vertices with the size of each block in the M-sequence.

Definition 6.26 (boundary,vicinity)**.** Let T be an acyclic bipartite tournament. Let M be any subset of vertices and π be a topological sort of T. Let $(X_1, Y_1, \dots)$ be the M-sequence of T. For any block $X_i \cup Y_i$, the set of vertices in X_i before the first M-vertex is called the left boundary of the block and the set of vertices in X_i after the last M-vertex is called the right boundary of the block. The vicinity of the block $X_i \cup Y_i$ is the union of the boundaries of $X_i \cup Y_i$, the right boundary of $X_{i-1} \cup Y_{i-1}$, Y_i and the left boundary of $X_{i+1} \cup Y_{i+1}$.

Lemma 6.27. *Let H be an M-homogeneous solution for a bipartite tournament T. Then, in the M-sequence $(X_1, Y_1, X_2, Y_2, \dots)$ of $T - H$, for each i, $\frac{|X_i|}{|X_i \cap M|} \leq 20 \log^3 k$ and $|Y_i| \leq 10 \log^3 k$. Further, there exists a topological sort of $T - H$ such that the size of each boundary of any block is at most $10 \log^3 k$ and the size of the vicinity of any block is at most $30 \log^3 k$.*

Proof. The lemma follows immediately after observing that the canonical sequence of $T - H$ is a refinement of M-sequence of $T - H$ and any topological sort of $T - H$ preserves the canonical sequence of $T - H$. □

Definition 6.28 (Back edge)**.** Let T be an M-consistent bipartite tournament for some $M \subseteq V(T)$ and $(X_1, Y_1, X_2, Y_2 \dots)$ be the M-sequence of T. An edge $u_i u_j \in E(T)$ is called a *back* edge if $u_i \in X_i \cup Y_i$, $u_j \in X_j \cup Y_j$ and $i - j \geq 1$. Furthermore, $u_i u_j$ is called *short back* edge if $i - j = 1$ and it is called *long back* edge if $i - j \geq 2$.

Lemma 6.29. *Any feedback vertex set disjoint from M must contain at least one end point of a long back edge.*

Proof. Let u_ju_i be a long back edge and $i < j$ and $u_i \in X_i, u_j \in X_j$. Then, there are two vertices $u_l \in X_l$ and $u_{l+1} \in X_{l+1}$ in M such that $i < l < l+1 < j$. This creates the cycle $u_iu_lu_{l+1}u_ju_i$. Since u_l and u_{l+1} are undeletable, the feedback vertex set must contain at least one of u_i and u_j.

Now, consider the case when $u_i \in Y_i$ and $u_j \in X_j$. Since u_i is (M, X_i)-conflicting, there is an out neighbor $u \in X_i$ of u_i. Again, since $j - i \geq 2$, there is a set X_l such that $i < l < j$ and we get a cycle $u_iuu_lu_ju$ where $u_l \in X_l$. The case when $u_i \in Y_i$ and $u_j \in Y_j$ is similar. □

As we know that in the M-sequence of $T - H$, there may be back edges. Since $T - H$ is acyclic, these edges do not participate in any cycle. We call them *simple* back edges. But, in the M-sequence of $T - P$, we may have back edges that form a cycle with two vertices of M and hence at least one end-point of these edges must belong to H. We call them *conflict* back edges. Hence, every back edge that is not a simple back edge is a conflict back edge. By Lemma 6.29, every long back edge is a conflict back edge. The M-homogeneity of H and Lemma 6.27 implies the following lemma.

Lemma 6.30. *Let H be an M-homogeneous solution for T. Then, there exists a permutation of $T - H$ such that the number of simple back edges between any consecutive blocks in the M-sequence of $T - H$ is at most $200 \log^6 k$.*

Hence, if in the M-sequence of $T - P$, there are more than $200 \log^6 k$ back edges between any consecutive pair of blocks, then we can branch on the choices of conflict back edges to be hit by H. The next definition and lemma captures this intuition.

Let T be a bipartite tournament such that $T - P$ is M-consistent for some sets $M, P \subseteq V(T)$. Let $\mathcal{L}'$ be a function such that given an M-consistent bipartite tournament T for some set $M \subseteq V(T)$ and an integer k, outputs the set of short back edges in the M-sequence $(X_1, Y_1, \dots)$ of T which is the union of all sets of back edges $E_{i,i+1}$ between $X_i \cup Y_i$ and $X_{i+1} \cup Y_{i+1}$ such that the size of matching in the bipartite graph $(X_i \cup Y_{i+1}, X_{i+1} \cup Y_i, E_{i,i+1})$ is at least $201 \log^8 k$. Let $\mathsf{long}(T, M, P)$ denote the set of long back edges in $T - P$.

Definition 6.31 (weakly-coupled)**.** An instance (T, M, P, F, k) of CFVS is said to be weakly-coupled if in the M-sequence of $T - P$, F is a subset of conflict back edges containing all long back edges such that the matching in back edges between any pair of consecutive blocks in $T - P - F$ is at most $201 \log^8 k$.

Since we can find a matching in bipartite graphs in polynomial time, it can be checked in polynomial time whether a given CFVS instance (T, M, P, F, k) is weakly-coupled or not.

Lemma 6.32. *There exists a γ-reduction from a CFVS instance $(T, M, P, \emptyset, k)$ to a family $\mathcal{C}_2 = \{(T, M, P, F_1, k), (T, M, P, F_2, k) \dots\}$ for $\gamma = 2^{O(\frac{k}{\log k})}$ such that every instance in $\mathcal{C}_2$ is weakly-coupled.*

Proof. We construct $\mathcal{C}_2$ as follows. For each $B \subseteq \mathcal{L}'(T-P, M, k)$ such that $|B| \leq \frac{2k}{\log^2 k}$ output a set $F_B := \mathcal{L}'(T, M, k) \setminus B \cup \mathsf{long}(T, M, P)$. For each set F_B, add the instance (T, M, P, F_B, k) in $\mathcal{C}_2$ if (T, M, P, F_B, k) weakly-coupled.

By the definition of γ-reduction and the construction of $\mathcal{C}_1$, the backward direction is trivial. Now we consider the forward direction. Let H be an M-homogeneous $(M, P, \emptyset, k)$-constrained solution for $(T, M, P, \emptyset, k)$ and $(X_1, Y_1, \dots)$ be the M-sequence of $T - P$. It is sufficient to show that there is a CFVS instance (T, M, P, F, k) such that H is a vertex cover of F. A pair of consecutive blocks is said to have *large back* edge matching if the size of a matching in the set of back edges between them is at least $201 \log^8 k$. Fix a permutation σ of $T - H$ and choose any permutation σ' of $T - P$ such that σ'_{T-H} is σ. By Lemma 6.30, we have that at most $200 \log^6 k$ edges are simple back edges between any pair of consecutive blocks in the M-sequence of $T - H$. Rest of the back edges are conflict back edges and must be hit by H. If the short back edge matching is large between a pair of blocks, then at least $201 \log^8 k - 200 \log^6 k \geq 200 \log^8 k$ of them are conflict back edges. Hence, the number of set pairs with large short edge matching can be at most $\frac{k}{200 \log^8 k}$. This implies that at most $2 \times \frac{k}{200 \log^8 k} \times 200 \log^6 k = \frac{2k}{\log^2 k}$ edges are simple back edges. Since the algorithm loops over all choices of subsets $B \subseteq \mathcal{L}'(T, M, k)$, $|B| \leq \frac{2k}{\log^2 k}$, $\mathcal{C}_2$ contains an instance with the required properties.

Moreover, $|\mathcal{C}_2|$ is bounded by the number of subsets B. Now $|\mathcal{L}'(T, M, k)| \leq |V(T)|^2$ which implies the number of subsets B is at most $(k^6)^{\frac{2k}{\log^2 k}} = 2^{6 \log k \times \frac{2k}{\log^2 k}} = 2^{O(\frac{k}{\log k})}$. Hence, $|\mathcal{C}_2| = 2^{O(\frac{k}{\log k})}$ □

Definition 6.33 (matched)**.** An instance (T, M, P, F, k) of CFVS is said to be matched if $F \cap E(T - P)$ forms a matching.

Note that it can be checked in polynomial time whether a given CFVS instance (T, M, P, F, k) is matched or not.

Lemma 6.34. *There exists a γ-reduction from a weakly-coupled CFVS instance (T, M, P, F, k) to $\mathcal{C}_3 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$ for $\gamma \leq 1.6181^k$ such that $\mathcal{C}_3$ is weakly-coupled and matched. In addition, for each $|P_i| = s \leq k$, $\mathcal{C}_3$ has at most 1.618^s CFVS instances.*

Proof. We construct the family $\mathcal{C}_3$ using a branching algorithm. Consider the graph $G := (V(T) \setminus P, F \cap E(T - P))$. Start with $k' = k$, $P' := P$ and $F' := E(G)$. In each branch node, the sets P', F' are updated and finally for each leaf node in the branch tree, the corresponding instance (T, M, P', F, k) is returned. As long as there is a vertex $v \in V(G)$ of degree at least 2 and $k' > 0$, branch by considering both the possibilities: $v \in H$ or $v \notin H$. In the branch in which v is picked, decrease k' by 1 and update $P' = P' \cup \{v\}$ and $F' = F' \setminus E(v)$. In the other branch, $N(v)$ is added to P', $E(N(v))$ is removed from F' and k' is decreased by $|E(N(v))|$. The algorithm stops branching further in a branch in which either $k' < 0$ or $k' > 0$ and for every vertex v, degree of v is at most 1. In the case that $k' < 0$ or $|F'| > k$, the algorithm terminates the branch without returning any

instance and moves on to other branches. Any returned instance (T, M, P', F, k) is added to $\mathcal{C}_3$ if the instance is regular, weakly-coupled and matched.

Again, the definition of the γ-reduction and above construction of $\mathcal{C}_3$, ensures the backward direction. Now we consider the forward direction. Let (T, M, P, F, k) be weakly-coupled and H be an M-homogeneous (M, P, F)-constrained solution of T. Since, the above branching algorithm adds an instance into $\mathcal{C}_3$ with P' containing P such that $F \cup E(T - P')$ forms a matching, if $\mathcal{C}_3$ is non-empty, all instances in it are weakly-coupled and matched. Since H hits F, there is a subset $\mathcal{P}''$ of H, such that F forms a matching in $T - (P \cup P'')$. Since the above algorithm via branching considers all possible subsets P' containing P that *make* F disjoint in some branch $P' \subseteq P$ implying that $\mathcal{C}_3$ contains an M-homogeneous (M, P', F)-constrained solution.

Now we argue about γ and the number of instances. Let s denote the size of P' in any instance (T, M, P', F, k). Since, H must hit F' and F' are disjoint $s \leq k$. As $P' \subseteq H$, $|F'| \leq k - s$. The recurrence relation for bounding the number of leaves with $|F'| = k - s$ in the branch tree of the above algorithm is given by:

$$g_s(k) \leq g_s(k-1) + g_s(k-2)$$

which solves to $g_s(k) \leq 1.618^s$ as $g_s(k) \leq 1$ for $k = s$. □

Definition 6.35 (LowBlockDegree)**.** An instance (T, M, P, F, k) of CFVS is said to be LowBlockDegree if in the M-sequence $(X_1, Y_1, X_2, Y_2, \dots)$ of $T - P$, $\mathsf{long}(T, M, P) \subseteq F$ and for every set $X_i \cup Y_i$, at most $201 \log^{10} k$ edges of $F \setminus E(T-P)$ are incident on $X_i \cup Y_i$.

Note that it can be checked in polynomial time whether a given CFVS instance (T, M, P, F, k) is LowBlockDegree or not.

Definition 6.36 (X-preferred vertex cover)**.** Given a bipartite graph G a set of vertices $X \subseteq V(T)$ and a set of edges $Q \subseteq E(G)$ such that Q is a matching in G, a minimum vertex cover C of Q is called X-vertex cover of Q if for every edge $e \in Q$ such that e has exactly one endpoint in X, C contains the endpoint of e in $V(G) \setminus X$.

Let $T := (A, B, E)$ be a bipartite tournament and let $X \subseteq A$. Let $\pi := (v_1, v_2, \dots, v_l)$ be a permutation of X. A vertex $v \in B$ is called inconsistent with π, if there is no index i such that every vertex in $\{v_1, v_2, \dots, v_i\}$ is an in-neighbor of v and every vertex in $\{v_{i+1}, v_{i+2,\dots,v_l}\}$ is an out-neighbor of v. Given a CFVS instance (T, M, P, F, k), a block in the M-sequence of $T - P$ is said to have large conflict edge matching if the block is incident with at least $201 \log^{10} k$ edges in $F_1 := F \cap E(T - P)$.

Lemma 6.37. *There exists a γ-reduction from a weakly-coupled and matched CFVS instance (T, M, P, F, k) to $\mathcal{C}_4 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$ for $\gamma = 2^{O(\frac{k}{\log k})}$ such that every instance in $\mathcal{C}_4$ is weakly-coupled, matched and LowBlockDegree.*

Proof. Using M, P, F, we construct $\mathcal{C}_4$. Start with the M-sequence of $T - P$. Let $n = |V(T)|$, $t = \frac{2k}{201 \log^{10} k}$, $P' := P$ and $F' := F \cap E(T - P)$. Branch on every family $\mathcal{B}$ of blocks such that $|\mathcal{B}| \leq t$. Branch on every subset M' of size at most $t \cdot 30 \log^3 k$. Let X be the union of M-sub-blocks and Y be the union of $\bar{M}$-sub-blocks in $\mathcal{B}$. Add every vertex in $Y \setminus M'$ to P'. Add every back edge neighbor of M' to P'. Branch on every permutation π of M'. Add every vertex of $X \setminus M'$ not consistent with the permutation π to P'. Let E' be the set of back edges incident on $X \setminus M'$. Let $G := (V(T), E')$. Note that G is a bipartite graph. Branch on every minimum vertex cover of G by adding it to P'. Add a $\bigcup \mathcal{B}$-preferred cover of conflict edges in F' incident on $(X \cup Y) \setminus P'$ to P'. Finally, we add a CFVS instance (T, M, P', F, k) to $\mathcal{C}_4$ if (T, M, P', F, k) is LowBlockDegree.

Correctness: First we show that $|\mathcal{C}_4| \leq \gamma$. $|\mathcal{C}_4|$ is bounded by the product of the number of family of blocks $\mathcal{B}$, the number of sets M', the number of permutations of M' and the number of minimum vertex cover of G. The number of family of blocks $\mathcal{B}$ is bounded by n^t as the number of blocks can be at most n. Similarly, the number of subsets M' is bounded by $n^{t \cdot 30 \log^3 k}$. The number of permutations is bounded by $(t \cdot 30 \log^3 k)!$. Since, (T, M, P, F, k) is weakly-coupled, the matching on back edges incident on any block is at most $201 \log^8 k$. Hence, the size of a maximum matching in G is at most $t \cdot 201 \log^8 k$. Hence, the number of minimal vertex cover of G is at most $2^{t \cdot 201 \log^8 k}$. Since, $n = |V(T)| = O(k^3)$, after little arithmetic manipulation, we have that $|\mathcal{C}_4| \leq n^t \times n^{t \cdot 30 \log^3 k} \times (t \cdot 30 \log^3 k)! \times 2^{t \cdot 201 \log^8 k} = 2^{O(\frac{k}{\log k})}$.

By the definition of the family $\mathcal{C}_4$ and of γ-reduction, the backward direction is immediate. For the forward direction, let (T, M, P, F, k) be a weakly-coupled and matched CFVS instance and let H be an M-homogeneous solution of (T, M, P, F, k). It is sufficient to show that $\mathcal{C}_4$ has an instance (T, M, P', F, k) such that $P' \subseteq H$.

Consider the M-sequence of $T - P$. Fix a permutation σ of $T - H$. Consider a permutation σ' of vertices in $T - P$ whose restriction to $T - H$ is σ. Let $\mathcal{B}$ be the family of blocks with *very large* matching in the set of conflict edges F'. Since $|H| \leq k$, the size of $\mathcal{B}$ is less than $t = \frac{2k}{201 \log^{10} k}$. Since the size of vicinity of any block is at most $30 \log^3 k$, at most $t \cdot 30 \log^3 k$ vertices form the vicinity M' of blocks in $\mathcal{B}$. Let X be the union of M-sub-blocks and Y be the union of $\bar{M}$-sub-blocks in $\mathcal{B}$. Then, vertices in $Y \setminus M'$ belong to H. Since, M' is the vicinity of the blocks, every back edge incident on M' is a conflict edge. Hence, the back edge neighbor of M' belongs to H. For the same reason, the set of back edges E' incident on $X \setminus M'$ are conflict edges and H contains a minimum cover of E'. As $M' \cap H = \emptyset$, any vertex inconsistent with $\sigma_{M'}$ also belongs to H. Now, every block in $\mathcal{B}$ is incident with conflict edges belong to F only which are disjoint, we can greedily include a vertex cover of these edges by preferring to pick the conflict edge neighbor of $\bigcup \mathcal{B}$ into H. This implies that every block in $\mathcal{B}$ after removing P' is not incident with any conflict edge and hence (T, M, P', F, k) is LowBlockDegree. Since P' includes all possibilities of the above choices, there is an instance (T, M, P', F, k) in $\mathcal{C}_4$ that satisfies the required properties. □

Definition 6.38 (Regular)**.** An instance (T, M, P, F, k) of CFVS is said to be

regular if in the M-sequence $(X_1, Y_1, X_2, Y_2, \dots)$ of $T - P$, for every set X_i of size at least $10\log^5 k$, there are at least $\frac{|X_i|}{10\log^5 k}$ vertices in M and $|Y_i| \leq 10\log^5 k$.

Note that it can be checked in polynomial time whether a given CFVS instance (T, M, P, F, k) is regular or not. Let $\mathcal{L}$ be a function such that given a CFVS instance (T, M, P, F, k) outputs the family of sets of vertices which is the union of all sets X_i and Y_j in the M-sequence of $T - P$ such that $\frac{|X_i|}{m_i} \geq 10\log^5 k$ where $m_i = |X_i \cap M|$ and $|Y_j| \geq 10\log^5 k$.

Lemma 6.39. *There exists a γ-reduction from a CFVS instance (T, M, P, F, k) to a family $\mathcal{C}_1 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$ of CFVS instances for $\gamma = 2^{O(\frac{k}{\log k})}$ such that every instance in $\mathcal{C}_1$ is regular.*

Proof. We construct $\mathcal{C}_1$ as follows. Compute the sets $\mathcal{L}(T, M, k)$. For each $B \subseteq \mathcal{L}(T, M, k)$ such that $|B| \leq \frac{2k}{\log^2 k}$, output a pair of sets $(M, P') = (M, P \cup \mathcal{L}(T, M, k) \setminus B)$. For each pair (M, P'), add a CFVS instance $(T, M, P', \emptyset, k)$ in $\mathcal{C}_1$ if $(T, M, P', \emptyset, k)$ is regular.

By the definition of γ-reduction and the construction of $\mathcal{C}_1$, the backward direction is trivial. For the forward direction, let H be an M-homogeneous (M, P, F)-CFVS solution of (T, M, P, F, k).

A set X_i in the M-sequence of $T - P$ is called *large* if the ratio $\frac{|X_i|}{m_i}$ is at least $10\log^5 k$. Similarly, Y_i is large if $|Y_i| \geq 10\log^5 k$. From each large set X_i at least $10m_i\log^5 k - 10m_i\log^3 k$ vertices belong to H. Similarly, from each large Y_i, at least $10\log^5 k - 10\log^3 k$ belongs to H. Hence, if t is the total number of M-vertices in the union of large sets, then in total at most $\frac{k}{10t\log^5 k - 10t\log^3 k} \times 10t\log^3 k \leq \frac{2k}{\log^2 k}$ vertices from the union of *large* sets in the M-sequence of $T - P$ do not belong to H. Since the algorithm loops over all choices of subsets $B \subseteq \mathcal{L}(T, M, k)$, $|B| \leq \frac{2k}{\log^2 k}$, $\mathcal{C}_1$ contains an instance (T, M, P', F, k) satisfying the required properties.

Moreover, $|\mathcal{C}_1|$ is bounded by the number of subsets B. Now $|\mathcal{L}(T, M, k)| \leq |V(T)|$ which implies the number of subsets B is at most $(k^3)^{\frac{2k}{\log^2 k}} = 2^{3\log k \times \frac{k}{\log^2 k}} = 2^{O(\frac{k}{\log k})}$. □

As noted before BTFVS instance (T, k) is equivalent to CFVS instance $(T, \emptyset, \emptyset, \emptyset, k)$, we combine the results in the above Lemmas (abusing the notation slightly).

Lemma 6.40. *There is a γ-reduction from a BTFVS instance (T, k) to a CFVS family $\mathcal{C}'$ for $\gamma \leq 1.6181^k$ such that every instance in $\mathcal{C}'$ is regular, weakly-coupled, matched and LowBlockDegree. In addition, for each $|P_2| = s \leq k$, $\mathcal{C}'$ has at most 1.618^s CFVS instances.*

Proof. Given the BTFVS instance (T, k), run the algorithm of Lemma 6.25 to obtain the CFVS family $\mathcal{C}$. For each instance $(T, M, P, \emptyset, k) \in \mathcal{C}$, run the algorithm of Lemma 6.39 to obtain the CFVS family $\mathcal{C}_1$. For each instance $(T, M, P, \emptyset, k) \in \mathcal{C}_1$, run the algorithm of Lemma 6.32 to obtain the CFVS family $\mathcal{C}_2$. For each instance $(T, M, P, F, k) \in \mathcal{C}_2$, run the algorithm of Lemma 6.34 to obtain the CFVS family

$\mathcal{C}_3$. For each instance $(T, M, P, F, k) \in \mathcal{C}_3$, run the algorithm of Lemma 6.37 to obtain the CFVS family $\mathcal{C}_5(T, M, P_2, F, k)$. $\mathcal{C}'$ is the union of these families.

The correctness and the running time follow from Lemmas 6.25, 6.39, 6.32, 6.34 and 6.37. □

We redefine the d-Feedback Vertex Cover defined in Section 5.4 with a slight modification. Let d, f and t be positive integers. Consider a class of mixed graphs $\mathcal{G}(d, f, t)$ in which each member is a mixed multigraph $\mathcal{T}$ with the vertex set $V(\mathcal{T})$ partitioned into vertex sets $V_1, V_2, \ldots, V_t$ and an undirected edge set $\mathcal{E}(\mathcal{T}) \subseteq \bigcup_{i<j} V_i \times V_j$ such that for each $i \in [t]$,

- $\mathcal{T}[V_i]$ is a bipartite tournament,
- the size of the feedback vertex set H_i for $\mathcal{T}[V_i]$ is at least f and at most $4f$,
- $\mathsf{deg}_{\mathcal{E}}(V_i) \leq d$.

Given a mixed multigraph $\mathcal{T} \in \mathcal{G}(d, f, t)$, a positive integer k, determine whether there exists a set $H \subseteq V(\mathcal{T})$ such that $|H| \leq k$ and $\mathcal{T} - H$ contains no undirected edges and is acyclic. If $\mathcal{E}(\mathcal{T})$ is disjoint, we call the problem as Disjoint Feedback Vertex Cover.

Lemma 6.41. *There exists a polynomial time algorithm that given a CFVS instance (T, M, P_2, F, k) that is regular, weakly-coupled, matched and LowBlockDegree outputs a partition $(V_1, V_2, \ldots, V_t)$ of $V(T) \setminus P$ such that $t \leq \frac{k}{201 \log^{12} k}$ and for each $i \in [t]$ V_i is a union of consecutive blocks in the M-sequence of $T - P_2$ and at least one of these hold*

- *the size of feedback vertex set of $T[V_i]$ is at least $f = 201 \log^{12} k$ and at most $804 \log^{12} k$,*
- *at least $200 \log^{12} k$ and at most $201 \log^{12} k$ edges in $F \cap E(T - P_2)$ are incident on V_i.*

Proof. Let $(X_1, Y_1 \ldots)$ be the M-sequence of $T - P_2$. Consider the sequence of blocks $(Z_1, Z_2 \ldots)$ such that for each i, $Z_i := X_i \cup Y_i$. Obtain the sequence $i_1 = 1 < i_2 < \ldots$ of indices such that $V_j := \bigcup_{i=i_j}^{i_{j+1}-1} Z_i$ as follows: for each j, keep including Z_i for $i \geq i_j$ into V_j and stop the moment at least one of the above conditions hold. To check the size of feedback vertex set in $T[V_j]$ use the approximation algorithm in Lemma 6.2 i.e. check if Lemma 6.2 outputs a feedback vertex set for $T[V_j]$ of size less than $4f$.

Since by regularity, the feedback vertex set of any block is at most $10 \log^5 k$ and since the CFVS instance is weakly-coupled, the size of a maximum matching on back edges between any consecutive blocks is at most $201 \log^8 k$. Since the CFVS instance is matched and LowBlockDegree, the size of maximum matching in conflict edges is at most $201 \log^{10} k$. Hence, including any block into a set V_i increases the size of the feedback vertex set of $T[V_i]$ by at most $10 \log^5 k + 201 \log^{10} k$. At the same time, the degree of V_i can increase by at most $201 \log^{10} k$. Hence, the above

algorithm outputs the required partition. Note that edges in $F \cap E(T - P_2)$ form a matching. Hence, $t \leq \frac{k}{201 \log^{12} k}$. □

Definition 6.42 (decoupled)**.** An instance (T, M, P, F, k) of CFVS is said to be decoupled if there is a partition $(V_1, V_2, \ldots, V_t)$ of $V(T) \setminus P$ such that $t \leq \frac{k}{201 \log^{12} k}$ and for each $i \in [t]$

- V_i is a union of consecutive blocks in the M-sequence of $T - P$,
- the size of feedback vertex set of $T[V_i]$ is at least $f = 201 \log^{12} k$ and at most $804 \log^{12} k$, or at least $200 \log^{12} k$ and at most $d = 201 \log^{12} k$ edges in $F \cap E(T - P)$ are incident on V_i.
- F contains short conflict edges between any pair of sets V_i and V_j.

Note that it can be checked in polynomial time whether a given CFVS instance (T, M, P, F, k) is decoupled or not.

Lemma 6.43. *There exists a γ-reduction from a regular, weakly-coupled, and matched CFVS instance (T, M, P_2, F, k) to a family $\mathcal{C}_6$ for $\gamma = 2^{O(\frac{k}{\log k})}$ such that every instance in $\mathcal{C}_6$ is regular, weakly-coupled, matched, LowBlockDegree and decoupled.*

Proof. Given (T, M, P_2, F, k), we construct the family $\mathcal{C}_6$. Using the algorithm of Lemma 6.41, we construct the partition $(V_1, V_2, \ldots, V_t)$ of $V(T) \setminus P_2$. For each V_i, let E_i be the set of back edges incident on V_i from $V(T) \setminus (P_2 \cup V_i)$. Let $J := \bigcup_{V_i} E_i$ be the union of such back edges. Now, we guess the subset B of back edges that are not hit by the required feedback vertex set. For every subset $B \subseteq J$ of size at most $2 \cdot t \cdot 200 \log^6 k$, let $J_B = J \setminus B$. We require that the feedback vertex set hits at least one end point of every edge in J_B. Let D be the vertex cover of J_B. For every subset $C \subseteq D$, define $P_C := C \cup N_{J_B}(D \setminus C)$. For each P_C, we add the CFVC instance (T, M, P_3, F, k) where $P_3 := P_2 \cup P_C$ into $\mathcal{C}_6$ if (T, M, P_3, F, k) is regular, weakly-coupled, matched, LowBlockDegree and decoupled.

The backward direction is trivial. For the forward direction, let H be an M-homogeneous (M, P, F)-CFVS solution. Observe that all the above algorithm does is consider all possibilities via which H may hit the back edges between $T[V_i \setminus P_2]$ and $T[V_i \setminus P_2]$ for any i, j. The number of choices of sets B is at most $(k^6)^{2 \cdot t \cdot 200 \log^6 k} = 2^{O(\frac{k}{\log k})}$. Note that in the M-sequence of $T - P_2$, the matching on short back edges between any pair of consecutive blocks is at most $201 \log^{10} k$. Hence, the vertex cover of these back edges is at most $201 \log^{10} k$. Since the number of sets in the partition $(V_1, V_2, \ldots)$ is at most $\frac{k}{f}$, the size of the total matching on short back edges J is at most $g = 201 \log^{10} k \times \frac{k}{f}$. Hence, the number of choices for C is at most $2^g = 2^{O(\frac{k}{\log k})}$. Hence, $\gamma = 2^{O(\frac{k}{\log k})} \times 2^{O(\frac{k}{\log k})} = 2^{O(\frac{k}{\log k})}$. □

Lemma 6.44. *There is a polynomial time reduction from a CFVS instance (T, M, P_2, F, k) that is regular, weakly-coupled, matched, LowBlockDegree and decoupled to an instance of* Disjoint Feedback Vertex Cover *$(\mathcal{T}, k')$ for $k' = k - |P_2|$.*

Proof. Given (T, M, P_2, F, k), construct the DFVS instance with vertex set $V(T) \setminus (M \cup P_2)$ and make the edges in $F \setminus E(T - P_2)$ between any two sets V_i and V_j undirected. For any solution H for (T, M, P_2, F, k), $H \setminus P_2$ is a feedback vertex set of $T - P_2$ that hits $F \setminus E(T - P_2)$. Hence, $H \setminus P_2$ is a feedback vertex cover for $(\mathcal{T}, k')$ for $k' = k - |P_2|$. In the backward direction, a solution S for $(\mathcal{T}, k')$ hits $F \setminus E(T - P_2)$ and is disjoint from M. Hence, $S \cup P_2$ is a solution for (T, M, P_2, F, k). □

At this point, we can use the following lemma from [KL16c] with the only difference being in the base case as we have a bipartite tournament instead of a *supertournament*. We replace the naive 3^k algorithm by 4^k algorithm to find a feedback vertex for each of $T[V_i]$. Note that bounding the size of feedback vertex set in each of $T[V_i]$ to $O(\log^{12} k)$ and the number of V_i's to at most $O(\frac{k}{\log^{12} k})$ implies that the maximum time spent in solving the base cases is at most $O(\frac{k}{\log^{12} k}) \cdot 2^{O(\log^{12} k)}$.

Lemma 6.45. *There exists an algorithm running in* $1.5874^s \cdot 2^{O(df \log k + d \log s)} \cdot n^{O(1)}$ *time which finds an optimal feedback vertex cover in a mixed multigraph* $\mathcal{T} \in \mathcal{G}(d, f, t)$ *in which the undirected edge set* $\mathcal{E}(\mathcal{T})$ *is disjoint and* $|\mathcal{E}(\mathcal{T})| = s$.

Theorem 19. *There exists an algorithm for* BTFVS *running in* $1.6181^k + n^{O(1)}$ *time.*

Proof. Using the algorithm of Lemma 6.40, construct the family $\mathcal{C}_5$ of CFVS instances. For each instance $(T, M, P_2, F, k) \in \mathcal{C}_5$, using the algorithm of Lemma 6.43 construct the family $\mathcal{C}_6$ of CFVS instances. Then for each CFVS instance (T, M, P, F, k) using Lemma 6.44, construct the DFVC instance $(\mathcal{T}, k - |P|)$ which is solved using the algorithm of Lemma 6.45. If for any instance, the algorithm of Lemma 6.45 outputs a solution set S of size at most $k - |P|$, then we output YES, otherwise output NO.

The correctness of the algorithm follows from the correctness of the algorithms in the Lemma 6.40, 6.43, 6.44 and 6.45. The running time of the algorithm is upper bounded by $\sum_{s=1}^{k} 1.618^{k-s} \times 1.5874^s \cdot 2^{O(df \log k + d \log s)} \cdot n^{O(1)} \leq 1.6181^k \cdot 2^{O(d^2 + d \log k)} \cdot n^{O(1)}$. □

Proposition 6.46. *[FGLS16] If there exists a parameterized algorithm for any vertex deletion problem into a hereditary graph class with running time* $c^k n^{O(1)}$*, then there exists an exact-exponential-time algorithm for the problem with running time* $(2 - \frac{1}{c})^{n+o(n)} n^{O(1)}$.

The above proposition immediately implies the following theorem.

Theorem 20. *There exists an algorithm for* BTFVS *running in* 1.3820^n *time.*

Chapter 7

Component Order Connectivity

7.1 Introduction

In the classic Vertex Cover problem, the input is a graph G and integer k, and the task is to determine whether there exists a vertex set S of size at most k such that every edge in G has at least one endpoint in S. Such a set is called a *vertex cover* of the input graph G. An equivalent definition of a vertex cover is that every connected component of $G - S$ has at most 1 vertex. This view of the Vertex Cover problem gives rise to a natural generalization: can we delete at most k vertices from G such that every connected component in the resulting graph has at most ℓ vertices? Here we study this generalization. Formally, for every integer $\ell \geq 1$, we consider the following problem, called ℓ-Component Order Connectivity (ℓ-COC).

ℓ-Component Order Connectivity (ℓ-COC)

Input:	A graph G and a positive integer k.
Question:	Does there exists a set $S \subseteq V(G)$ such that $\|S\| \leq k$ and the maximum size of a component in $G - S$ is at most ℓ?

The set S is called an ℓ-COC *solution*. For $\ell = 1$, ℓ-COC is just the Vertex Cover problem. Aside from being a natural generalization of Vertex Cover, the family $\{\ell\text{-COC} : \ell \geq 1\}$ of problems can be thought of as a vulnerability measure of the graph G - how many vertices of G have to fail for the graph to break into small connected components? For a study of ℓ-COC from this perspective see the survey of Gross et al. [GHI+13].

From the work of Lewis and Yannakakis [LY80b] it immediately follows that ℓ-COC is NP-complete for every $\ell \geq 1$. This motivates the study of ℓ-COC within paradigms for coping with NP-hardness, such as approximation algorithms [WS11], exact exponential time algorithms [FK10], parameterized algorithms [CFK+15, DF13] and kernelization [Kra14, LMS12]. The ℓ-COC problems have (for some values of ℓ) been studied within all four paradigms, see the related work section.

In this work we focus on ℓ-COC from the perspective of parameterized complexity and kernelization. Our main result is an algorithm that given an instance (G, k)

of ℓ-COC, runs in polynomial time, and outputs an equivalent instance (G', k') such that $k' \leq k$ and $|V(G')| \leq 2\ell k$. This is called a *kernel* for ℓ-COC with $2\ell k$ vertices. Our kernel significantly improves over the previously best known kernel with $O(\ell k(k + \ell))$ vertices by Drange et al. [DDvtH14]. Indeed, for $\ell = 1$ our kernel matches the size of the smallest known kernel for Vertex Cover [CKJ01] that is based on the classic theorem of Nemhauser and Trotter [NJ74].

Related Work. 1-COC, better known as Vertex Cover, is extremely well studied from the perspective of approximation algorithms [WS11, DS05], exact exponential time algorithms [FGK09, Rob86, XN13], parameterized algorithms [CFK+15, CKX10] and kernelization [CKJ01, NJ74]. The kernel with $2k$ vertices for Vertex Cover is considered one of the fundamental results in the field of kernelization. The 2-COC problem is also well studied, and has been considered under several different names. The problem, or rather the dual problem of finding a largest possible set S that induces a subgraph in which every connected component has order at most 2, was first defined by Yannakakis [Yan81] under the name Dissociation Set. The problem has attracted attention in exact exponential time algorithms [KKS11, XK15], the fastest currently known algorithm [XK15] has running time $O(1.3659^n)$. 2-COC has also been studied from the perspective of parameterized algorithms [CCH+16, Tu15] (under the name Vertex Cover P_3) as well as approximation algorithms [TZ11]. The fastest known parameterized algorithm, due to Chang et al. [CCH+16] has running time $1.7485^k n^{O(1)}$, while the best approximation algorithm, due to Tu and Zhou [TZ11] has factor 2.

For the general case of ℓ-COC, $\ell \geq 1$, Drange et al. [DDvtH14] gave a simple parameterized algorithm with running time $(\ell + 1)^k n^{O(1)}$, and a kernel with $O(k\ell(\ell + k))$ vertices. The parameterized algorithm of Drange et al. [DDvtH14] can be improved to $(\ell + 0.0755)^k n^{O(1)}$ by reducing to the $(\ell + 1)$-Hitting Set problem, and applying the iterative compression based algorithm for $(\ell + 1)$-Hitting Set due to Fomin et al. [FGK+10]. The reduction to $(\ell + 1)$-Hitting Set, coupled with the simple factor $(\ell + 1)$-approximation algorithm for $(\ell + 1)$-Hitting Set [WS11] immediately also yields an $(\ell + 1)$-approximation algorithm for ℓ-COC. There has also been some work on ℓ-COC when the input graph is restricted to belong to a graph class, for a discussion of this work see [DDvtH14].

Comparing the existing results with our work, we see that our kernel improves over the kernel of Drange et al. [DDvtH14] from at most $O(k\ell(\ell + k))$ vertices to at most $2k\ell$ vertices. Our kernel is also the first kernel with a linear number of vertices for every fixed $\ell \geq 2$.

Overview of the algorithm. Our kernel for ℓ-COC hinges on the concept of a *reducible pair* of vertex sets. Essentially (*this is not the formal definition used in the chapter!*), a reducible pair is a pair (X, Y) of disjoint subsets of $V(G)$ such that $N(Y) \subseteq X$, every connected component of $G[Y]$ has size at most ℓ, and every solution S to G has to contain at least $|X|$ vertices from $G[X \cup Y]$. If a reducible pair is identified, it is easy to see that one might just as well pick all of X into the solution S, since any solution has to pay $|X|$ inside $G[X \cup Y]$, and after X is deleted, Y breaks down into components of size at most ℓ and is completely

eliminated from the graph.

At this point there are several questions. (a) How does one argue that a reducible pair is in fact reducible? That is, how can we prove that any solution has to contain at least $|X|$ vertices from $X \cup Y$? (b) How big does G have to be compared to k before we can assert the existence of a reducible pair? Finally, (c) even if we can assert that G contains a reducible pair, how can we find one in polynomial time?

To answer (a) we restrict ourselves to reducible pairs with the additional property that each connected component C of $G[Y]$ can be assigned to a vertex $x \in N(C)$, such that for every $x \in X$ the total size of the components assigned to x is at least ℓ. Then x together with the components assigned to it form a set of size at least $\ell + 1$ and have to contain a vertex from the solution. Since we obtain such a connected set for each $x \in X$, the solution has to contain at least $|X|$ vertices from $X \cup Y$. Again we remark that this definition of a reducible pair is local to this section, and not the one we actually end up using.

To answer (b) we first try to use the q-Expansion Lemma (see [CFK$^+$15]), a tool that has found many uses in kernelization. Roughly speaking the Expansion Lemma says the following: if $q \geq 1$ is an integer and H is a bipartite graph with bipartition (A, B) and B is at least q times larger than A, then one can find a subset X of A and a subset Y of B such that $N(Y) \subseteq X$, and an assignment of each vertex $y \in Y$ to a neighbor x of y, such that every vertex x in X has at least q vertices in Y assigned to it.

Suppose now that the graph does have an ℓ-COC solution S of size at most k, and that $V(G) \setminus S$ is sufficiently large compared to S. The idea is to apply the Expansion Lemma to the bipartite graph H, where the A side of the bipartition is S and the B side has one vertex for each connected component of $G - S$. We put an edge in H between a vertex v in S and a vertex corresponding to a component C of $G - S$ if there is an edge between v and C in G. If $G - S$ has at least $|S| \cdot \ell$ connected components, we can apply the ℓ-Expansion Lemma on H, and obtain a set $X \subseteq S$, and a collection $\mathcal{Y}$ of connected components of $G - X$ satisfying the following properties. Every component $C \in \mathcal{Y}$ satisfies $N(C) \subseteq X$ and $|C| \leq \ell$. Furthermore, there exists an assignment of each connected component C to a vertex $x \in N(C)$, such that every $x \in X$ has at least ℓ components assigned to it. Since x has at least ℓ components assigned to it, the total size of the components assigned to x is at least ℓ. But then, X and $Y = \bigcup_{C \in \mathcal{Y}} C$ form a reducible pair, giving an answer to question (b). Indeed, this argument can be applied whenever the number of components of $G - S$ is at least $\ell \cdot |S|$. Since each component of $G - S$ has size at most ℓ, this means that the argument can be applied whenever $|V(G) \setminus S| \geq \ell^2 \cdot |S| \geq \ell^2 k$.

Clearly this argument fails to yield a kernel of size $2\ell k$, because it is only applicable when $|V(G)| = \Omega(\ell^2 k)$. At this point we note that the argument above is extremely wasteful in one particular spot: we used the *number* of components assigned to x to lower bound the *total size* of the components assigned to x. To avoid being wasteful, we prove a new variant of the Expansion Lemma, where the vertices on the B side of the bipartite graph H have non-negative integer

weights. This new Weighted Expansion lemma states that if $q, W \geq 1$ are integers, H is a bipartite graph with bipartition (A, B), every vertex in B has a non-negative integer weight which is at most W, and the total weight of B is at least $(q + W - 1) \cdot |A|$, then one can find a subset X of A and a subset Y of B such that $N(Y) \subseteq X$, and an assignment of each vertex $y \in Y$ to a neighbor x of y, such that for every vertex in X, the total weight of the vertices assigned to it is at least q. The proof of the Weighted Expansion Lemma is based on a combination of the usual, unweighted Expansion Lemma with a variant of an argument by Bezáková and Dani [BD05] to round the linear program for Max-min Allocation of goods to customers.

Having the Weighted Expansion Lemma at hand we can now repeat the argument above for proving the existence of a reducible pair, but this time, when we build H, we can give the vertex corresponding to a component C of $G - S$ weight $|C|$, and apply the Weighted Expansion Lemma with $q = \ell$ and $W = \ell$. Going through the argument again, it is easy to verify that this time the existence of a reducible pair is guaranteed whenever $|V(G) \setminus S| \leq (2\ell - 1)k$, that is when $|V(G)| \geq 2\ell k$.

We are now left with question (c) - the issue of how to *find* a reducible pair in polynomial time. Indeed, the proof of existence crucially relies on the knowledge of an (optimal) solution S. To find a reducible pair we use the linear programming relaxation of the ℓ-COC problem. We prove that an optimal solution to the LP-relaxation has to highlight every reducible pair (X, Y), essentially by always setting all the variables corresponding to X to 1 and the variables corresponding to Y to 0. For Vertex Cover (i.e 1-COC), the classic Nemhauser Trotter Theorem [NJ74] implies that we may simply include all the vertices whose LP variable is set to 1 into the solution S. For ℓ-COC with $\ell \geq 2$ we are unable to prove the corresponding statement. We are however, able to prove that if a reducible pair (X, Y) exists, then X (essentially) has to be assigned 1 and Y (essentially) has to be assigned 0. We then give a polynomial time algorithm that extracts X and Y from the vertices assigned 1 and 0 respectively by the optimal linear programming solution. Together, the arguments (b) and (c) yield the kernel with $2\ell k$ vertices. We remark that to the best of our knowledge, after the kernel for Vertex Cover [CKJ01] our kernel is the first example of a kernelization algorithm based on linear programming relaxations.

Overview of the chapter. The kernel for ℓ-COC is proved in Sections 7.2, 7.3 and 7.4. In Section 7.2 we prove the necessary adjustment of the results on Max-Min allocation of Bezáková and Dani [BD05] that is suitable to our needs. In Section 7.3 we state and prove our new Weighted Expansion Lemma, and in Section 7.4 we combine all our results to obtain the kernel.

7.2 Max-min Allocation

We will now view a bipartite graph $G := ((A, B), E)$ as a relationship between "customers" represented by the vertices in A and "items" represented by the

vertices in B. If the graph is supplied with two functions $w_a : A \to \mathbb{N}$ and $w_b : B \to \mathbb{N}$, we treat these functions as a "demand function" and a "capacity" function, respectively. That is, we consider each item $v \in B$ to have value $w_b(v)$, and every customer $u \in A$ wants to be assigned items worth at least $w_a(u)$. An edge between $u \in A$ and $v \in B$ means that the item v can be given to u.

A weight function $f : E(G) \to \mathbb{N}$ describes an assignment of items to customers, provided that the items can be "divided" into pieces and the pieces can be distributed to different customers. However this "division" should not create more value than the original value of the items. Formally we say that the weight function *satisfies* the capacity constraint $w_b(v)$ of $v \in B$ if $\sum_{uv \in E(G)} f(uv) \leq w_b(v)$. The weight function satisfies the capacity constraints if it satisfies the capacity constraints of all items $v \in B$.

For each item $u \in A$, we say that f *allocates* $\sum_{uv \in E(G)} f(uv)$ value to u. The weight function f *satisfies* the demand $w_a(u)$ of $u \in A$ if it allocates at least $w_a(u)$ value to u, and f satisfies the demand constraints if it does so for all $u \in A$. In other words, the weight function satisfies the demands if every customer gets items worth at least her demand. The weight function f *over-satisfies* a demand constraint $w_a(u)$ of u if it allocates strictly more than $w_a(u)$ to u.

We will also be concerned with the case where items are indivisible. In particular we say that a weight function $f : E(G) \to \mathbb{N}$ is *unsplitting* if for every $v \in B$ there is at most one edge $uv \in E(G)$ such that $f(uv) > 0$. The essence of the next few lemmas is that if we have a (splitting) weight function f of items whose value is at most W, and f satisfies the capacity and demand constraints, then we can obtain in polynomial-time an unsplitting weight function f' that satisfies the capacity constraints and violates the demand constraints by at most $(W - 1)$. In other words we can make a splitting distribution of items unsplitting at the cost of making each customer lose approximately the value of the most expensive item.

Allocating items to customers in such a way as to maximize satisfaction is well studied in the literature. The lemmata 7.1 and 7.2 are very similar, both in statement and proof, to the work of Bezáková and Dani [BD05][Theorem 3.2], who themselves are inspired by Lenstra et al. [LST90]. However we do not see a way to directly use the results of Bezáková and Dani [BD05], because we need a slight strengthening of (a special case of) their statement.

Lemma 7.1. *There exists a polynomial-time algorithm that given a bipartite graph G, a capacity function $w_b : B \to \mathbb{N}$, a demand function $w_a : A \to \mathbb{N}$ and a weight function $f : E(G) \to \mathbb{N}$ that satisfies the capacity and demand constraints, outputs a function $f' : E(G) \to \mathbb{N}$ such that f' satisfies the capacity and demand constraints and the graph $G_{f'} = (V(G), \{uv \in E(G) \mid f'(uv) > 0\})$ induced on the non-zero weight edges of G is a forest.*

Proof. We start with f and in polynomially many steps, change f into the required function f'. If $G_f = (V(G), \{uv \in E(G) \mid f(uv) > 0\})$ is a forest, then we return $f' = f$. Otherwise, suppose that G_f contains a cycle $C := e_1 e_2 e_3 \ldots e_{2s}$. Proceed as follows. Without loss of generality, suppose $c = f(e_1) = min\{f(e) \mid e \in C\}$, and note that $c > 0$. Compute the edge weight function $f^\star : E \to \mathbb{R}$ defined as follows.

For $e_i \in C$, we define $f^\star(e_i) = f(e) - c$ if i is odd, and define $f^\star(e_i) = f(e) + c$ if i is even. For $e \notin C$ we define $f^\star(e_i) = f(e)$.

Every vertex of G is incident to either 0 or exactly 2 edges of C. If the vertex v is incident to two edges of C then one of these edges, say e_{2i}, has even index in C, and the other, e_{2i+1} has odd. For the edge e_{2i} we have $f^\star(e_{2i}) = f(e_{2i}) + c$ and for e_{2i+1} we have $f^\star(e_{2i+1}) = f(e_{2i+1}) - c$. Thus we conclude that for all $v \in V(G)$, $\sum_{u \in N(v)} f^\star(uv) = \sum_{u \in N(v)} f(uv)$, and that therefore $f^\star$ satisfies the capacity and demand constraints. Furthermore at least one edge that is assigned non-zero weight by f is assigned 0 by $f^\star$ and $G_{f^\star} = (V(G), \{uv \in E(G) \mid f^\star(uv) > 0\})$ has one less cycle than G_f. For a polynomial-time algorithm, repeatedly apply the process described above to reduce the number of edges with non-zero weight, as long as $G_{f^\star}$ contains a cycle. □

Lemma 7.2. *There exists a polynomial-time algorithm with the following specifications. It takes as input a bipartite graph $G := ((A, B), E)$, a demand function $w_a : A \to \mathbb{N}$, a capacity function $w_b : B \to \mathbb{N}$, an edge weight function $f : E(G) \to \mathbb{N}$ that satisfies both the capacity and demand constraints, and a vertex $r \in A$. The algorithm outputs an unsplitting edge weight function $h : E(G) \to \mathbb{N}$ that satisfies the capacity constraints, satisfies the demands $w'_a = w_a - (W - 1)$ where $W = \max_{v \in B} w_b(v)$, and additionally satisfies the demand $w_a(r)$ of r.*

Proof. Without loss of generality the graph $G_f := (V(G), \{uv \in E(G) \mid f(uv) > 0\})$ is a forest. If it is not, we may apply Lemma 7.1 to f, and obtain a function f' that satisfies the capacity and demand constraints, and such that $G_{f'} = (V(G), \{uv \in E(G) \mid f'(uv) > 0\})$ is a forest. We then rename f' to f. By picking a root in each connected component of G_f we may consider G_f as a rooted forest. We pick the roots as follows, if the component contains the special vertex r, we pick r as root. If the component does not contain r, but contains at least one vertex $u \in A$, we pick that vertex as the root. If the component does not contain any vertices of A then it does not contain any edges and is therefore a single vertex in B, we pick that vertex as root. Thus, every item $v \in B$ that is incident to at least one edge in G_f has a unique *parent* $u \in A$ in the forest G_f. We define the new weight function h. For every edge $uv \in E(G)$ with $u \in A$ and $v \in B$ we define $h(uv)$ as follows. $h(uv) = w_b(v)$ if u is the parent of v in G_f, and $h(uv) = 0$ otherwise.

Clearly h is unsplitting and satisfies the capacity constraints. We now prove that h also satisfies the demand constraints w'_a and satisfies the demand constraint $w_a(r)$ of r. Consider the demand constraint $w'_a(u)$ for an arbitrary customer $u \in A$. There are two cases, either u is the root of the component of G_f or it is not. If u is the root, then for every edge $uv \in E(G)$ such that $f(uv) > 0$ we have that $uv \in E(G_f)$ and consequently that u is the parent of v. Hence $h(uv) = w_b(v) \geq f(uv)$, and therefore h satisfies the demand $w_a(u)$ of u. Since $w_a(u) \geq w'_a(u)$, we have that h satisfies the demand $w'_a(u)$. Furthermore, since r is the root of its component this also proves that h satisfies the demand $w_a(r)$.

Consider now the case that u is not the root of its component in G_f. Then u has a unique parent in G_f, call this vertex $v^\star \in B$. We first prove that

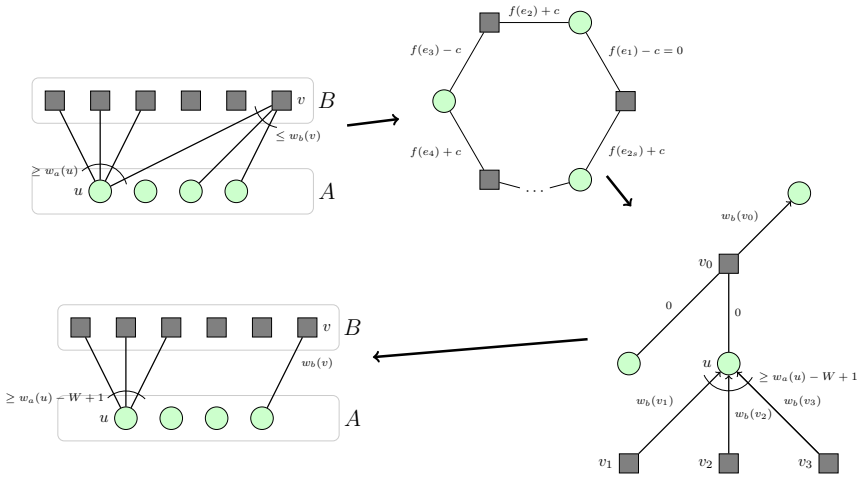

Figure 7.1: Proof of Lemma 7.1 and 7.2. Cyclically shift smallest weight in a non-zero weight cycle to obtain a forest. Root each tree in the forest at a vertex in A such that each vertex in B has a parent in A. Assign the value of $v \in B$ to its parent $u \in A$. In this new assignment, a non-root vertex $u \in A$ *loses* its parent $v_0 \in B$ and $f(v_0u) \leq W-1$ which explains the cost of making a splitting assignment into an unsplitting assignment.

$f(uv^\star) \leq w_b(v^\star) - 1$. Indeed, since $v^\star$ is incident to the edge $uv^\star$ we have that $v^\star$ has a parent $u^\star$ in G_f, and that $u^\star \neq u$ because $v^\star$ is the parent of u. We have that $f(u^\star v^\star) + f(uv^\star) \leq w_b(v^\star)$ and that $f(u^\star v^\star) \geq 1$, because $u^\star v^\star$ is an edge in G_f. It follows that $f(uv^\star) \leq w_b(v^\star) - 1$. We now proceed to proving that h satisfies the demand $w'_a(u)$.

For every edge $uv \in E(G) \setminus \{uv^\star\}$ such that $uv \in E(G)$ such that $f(uv) > 0$ we have that $uv \in E(G_f)$ and consequently that u is the parent of v. Hence we have that $h(uv) = w_b(v) \geq f(uv)$. Furthermore $h(uv^\star) = 0$ while $f(uv^\star) \leq w_b(v^\star) - 1 \leq W - 1$. Therefore h satisfies the demand $w'_a(u)$. □

7.3 The Weighted Expansion Lemma

Our kernelization algorithm will use "q-expansions" in bipartite graphs, a well known tool in kernelization [CFK+15]. We begin by stating the definition of a q-expansion and review the facts about them that we will use.

Definition 7.3 (**q-expansion**)**.** Let $G := ((A, B), E)$ be a bipartite graph. We say that A has q-expansion into B if there is a family of sets $\{V_a \mid V_a \subseteq N(a), |V_a| \geq q, a \in A\}$ such that for any pair of vertices $a_i, a_j \in A$,$i \neq j$, $V_{a_i} \cap V_{a_j} = \emptyset$.

Definition 7.4 (**Twin graph**)**.** For a bipartite graph $G := ((A, B), E)$ with a weight function $w_b : B \to \mathbb{N}$, the twin graph $T_{AB} := (A, B')$ of G is obtained as follows: B' contains $|w_b(v)|$ twins of every vertex $v \in B$ i.e. $B' := \{v_1, v_2, \ldots v_{w_b(v)} \mid v \in B\}$ and edges in T_{AB} such that for all $v \in B$ and $i \in [w_b(v)], N(v_i) = N(v)$ i.e. $E(T_{AB}) := \{av_i | a \in A, v_i \in B', v \in B, av \in E(G)\}$.

Lemma 7.5. *[CFK+15] Let G be a bipartite graph with bipartition (A, B). Then there is a q-expansion from A into B if and only if $|N(X)| \geq q|X|$ for every $X \subseteq A$. Furthermore, if there is no q-expansion from A into B, then a set $X \subseteq A$ with $|N(X)| < q|X|$ can be found in polynomial-time.*

Lemma 7.6 (Expansion Lemma [CFK+15])**.** *Let $q \geq 1$ be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$, and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there is a q-expansion of X into Y, and no vertex in Y has a neighbor outside X, i.e. $N(Y) \subseteq X$. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Lemma 7.7 (folklore)**.** *There exists a polynomial-time algorithm that given a bipartite graph $G := ((A, B), E)$ and an integer q decides (and outputs in case yes) if there exist sets $X \subseteq A, Y \subseteq B$ such that there is a q-expansion of X into Y.*

Proof. We describe a recursive algorithm. If $A = \emptyset$ or $B = \emptyset$, then output NO and terminate. Otherwise, construct the twin graph T_{BA} with weight function $w : A \to \mathbb{N}$ where for all $u \in A, w(u) = q$ and let M be a maximum matching in T_{BA}. Consider the graph $G' := (A, B)$ with edge set $E(G') := \{uv, u \in A, v \in B \mid u_i v \in M\}$. Let $A' \subseteq A$ such that for all $u \in A', d_{G'}(u) \geq q$ and let $B' \subseteq B$ such that $B' := \bigcup_{u \in A'} N_{G'}(u)$. If $N(B') \subseteq A'$, then return (A', B') and terminate. Otherwise, recurse on $G[A' \cup (B \setminus N_G(A \setminus A'))]$.

If there are no sets X, Y such that there is a q-expansion of X into Y, then for any pair of sets $A' \subseteq A, B' \subseteq B$ either $N(B') \setminus A' \neq \emptyset$ or $|B'| < q|A'|$. Since at each recursive step, the size of the graph with which the algorithm calls itself decreases, eventually either A' becomes empty or $B \setminus N_G(A \setminus A')$ becomes empty. Hence, the algorithm outputs no. Now we need to show that if there exist sets (A^*, B^*) such that there is a q-expansion of A^* into B^*, then at each recursive call, we have that $A^* \subseteq A$ and $B^* \subseteq B$. At the start of the algorithm, $A^* \subseteq A$ and $B^* \subseteq B$. Since $N(B^*) \subseteq A^*$ and for all $u \in A^*$ $d_G(u) \geq q$, we have that $A^* \cup B^* \subseteq V(G')$. If $N(B') \subseteq A'$, then the algorithm of Lemma 7.6 when run on G', q will output (A^*, B^*). Note that $B^* \subseteq B'$. At the recursive step, $A^* \subseteq A'$ and since $B^* \cap N_G(A \setminus A') = \emptyset$, we have that $B^* \subseteq B' \setminus N_G(A \setminus A')$. Hence, $G[A^* \cup B^*]$ is a subgraph of $G[A' \cup (B \setminus N_G(A \setminus A'))]$ which concludes the correctness of the algorithm. Since at each recursive call the size of the graph decreases by at least 1, the total time taken by the above algorithm is polynomial in n. □

One may think of a q-expansion in a bipartite graph with bipartition (A, B) as an allocation of the items in B to each customer in A such that every customer gets at least q items. For our kernel we will need a generalization of q-expansions to the setting where the items in B have different values, and every customer gets items of total value at least q.

Definition 7.8 (**Weighted q-expansion**)**.** Let $G := ((A, B), E)$ be a bipartite graph with capacity function $w_b : B \to \mathbb{N}$. Then, a weighted q-expansion in G is an edge weight function $f : E(G) \to \mathbb{N}$ that satisfies the capacity constraints

w_b and also satisfies the demand constraints $w_a = q$. For an integer $W \in \mathbb{N}$, the q-expansion f is called a *W-strict* q-expansion if f allocates at least $q + W - 1$ value to at least one vertex r in A, and in this case we say that f is W-strict at r. Further, a q-expansion f is *strict* (at r) if it is 1-strict (at r). If f is unsplitting we call f an *unsplitting* q-expansion.

Lemma 7.9. *There exists a polynomial-time algorithm that given a bipartite graph $G := ((A, B), E)$, an integer q and a capacity function $w_b : B \to \mathbb{N}$ outputs (if it exist) two sets $X \subseteq A$ and $Y \subseteq B$ along with a weighted q-expansion in $G[X \cup Y]$ such that $N(Y) \subseteq X$.*

Proof. Construct the twin graph $T_{AB} := (A, B')$ of G. Run the algorithm of Lemma 7.7 with input T_{AB}, q that outputs sets $X \subseteq A$ and $Y' \subseteq B'$ such that X has q-expansion into Y' and $N(Y') \subseteq X$. Consider the set $Y := \{v \in B \mid v_i \in Y'\}$. Define a weight function $f : E(G[X \cup Y]) \to \mathbb{N}$ as follows: for all $uv \in E(G[X \cup Y])$ $f(uv) = |\{v_i \in Y' | v_i \text{ matched to } u\}|$.

Clearly, $N(Y) \subseteq X$. Now we claim that f is a weighted q-expansion in $G[X \cup Y]$ with capacity function w_b and demand function $w_a = q$. For any vertex $u \in A$, there are at least q vertices in Y' are matched to u. Hence for all $u \in A$, we have that $\sum_{v \in N(u)} f(uv) \geq q = w_a$. At the same time, for any vertex $v \in B$, there are at most $w_b(v)$ copies of v in Y'. Therefore, for all $v \in Y$ we have $\sum_{u \in N(v)} f(uv) \leq w_b(v)$. □

Lemma 7.10. *There exists a polynomial-time algorithm that given a weighted q-expansion $f : E(G) \to \mathbb{N}$ in $G := ((A, B), E)$, a capacity function $w_b : B \to \mathbb{N}$ and an integer W such that $W = \max_{e \in E(G)} f(e)$ outputs an unsplitting W-strict weighted $(q - W + 1)$-expansion in G.*

Proof. Run the algorithm of Lemma 7.2 with inputs $G, f, w_a = q, w_b, W$ and a vertex $u \in A$. In case f is strict, u is the vertex r that makes f strict. Let the function $h : E(G) \to \mathbb{N}$ be the output of Lemma 7.2. Now h is an unsplitting edge weight function that satisfies the capacity constraints, satisfies the demands $q - W + 1$, and additionally satisfies the demand q of u. Hence, h is the required unsplitting weighted W-strict $(q - W + 1)$-expansion in G. □

Lemma 7.11 (Weighted Expansion Lemma)**.** *Let $q, W \geq 1$ be positive integers and G be a bipartite graph with vertex bipartition (A, B) and $w_b : B \to \{1, \ldots, W\}$ be a capacity function such that $\sum_{v \in B} w_b(v) \geq (q + W - 1) \cdot |A|$, and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that $N(Y) \subseteq X$ and there is an unsplitting weighted W-strict q-expansion of X into Y. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Proof. Construct the twin graph T_{AB} from G and w_b, the bipartition of T_{AB} is (A, B'). Now, obtain using the Expansion Lemma 7.6 with $q' = q + W - 1$ on T_{AB} sets $X \subseteq A$ and $Y' \subseteq B'$, such that $N(Y') \subseteq X$ and there is a $(q+W-1)$-expansion from X to Y' in T_{AB}.

Let $Y := \{v \in B \mid v_i \in Y'\}$ (here the $v_i \in Y'$ are as in Definition 7.4). Then $N(Y) \subseteq X$ and the $(q+W-1)$-expansion from X to Y' in T_{AB} immediately yields a weighted $(q+W-1)$-expansion f from X to Y in G. Applying Lemma 7.10 on $G[X \cup Y]$ using the weighted $(q+W-1)$-expansion f proves the statement of the lemma. □

7.4 Obtaining the Linear Kernel

Definition 7.12. For a graph G and a pair of vertex-disjoint sets $X, Y \subseteq V(G)$, we define the weighted graph $\tilde{G}_{XY}$ as follows: $V(\tilde{G}_{XY}) := X \cup \tilde{Y}$ such that there is a bijection $h : cc(G[Y]) \to \tilde{Y}$ where $cc(G[Y])$ is the set of connected components of $G[Y]$. $E(\tilde{G}_{XY}) := \{xc \mid x \in X, c \in \tilde{Y}, c = h(C) \text{ and } x \in N_G(C)\}$. We also define a weight function $w : \tilde{Y} \to \mathbb{N}$ such that for all $c \in \tilde{Y}, w(c) = |h^{-1}(c)|$.

Definition 7.13 (Reducible Pair). For a graph G, a pair of vertex-disjoint sets (X, Y) where $X, Y \subseteq V(G)$ is called a (strict) reducible pair if $N(Y) \subseteq X$, the size of every component in $G[Y]$ is at most ℓ, and there exists a (strict) weighted $(2\ell-1)$-expansion in $\tilde{G}_{XY}$.

Definition 7.14. A reducible pair (X, Y) is called minimal if there is no reducible pair (X', Y') such that $X' \subset X$ and $Y' \subseteq Y$.

Lemma 7.15. *There exists a polynomial-time algorithm that given an ℓ-COC instance (G, k) together with a vertex-disjoint set pair $A, B \subseteq V(G)$ outputs (if it exists) a reducible pair (X, Y) where $X \subseteq A$ and $Y \subseteq B$.*

Proof. Construct $\tilde{G}_{AB} := (A, \tilde{B})$ and run the algorithm of Lemma 7.9 with input $\tilde{G}_{AB}, w, q = 2\ell - 1$ which outputs sets $X \subseteq A$ and $Y' \subseteq \tilde{B}$ (if it exists) along with a weighted $(2\ell-1)$-expansion of X into Y' such that $N(Y') \subseteq X$. Now from Y' we obtain the set $Y := \bigcup_{y \in Y'} h^{-1}(y)$. Clearly, $N(Y) \subseteq X$ and hence, (X, Y) is the desired reducible pair. □

Lemma 7.16. *Given an ℓ-COC instance (G, k), if $|V(G)| \geq 2\ell k$ and (G, k) is a* YES*-instance, then there exists a reducible pair (X, Y).*

Proof. Without loss of generality, we can assume that G is a connected graph. Let S be an ℓ-COC solution of size at most k. Clearly, $|V \setminus S| \geq (2\ell-1)k$. We define $A := S$ and $B := V \setminus S$ and construct $\tilde{G}_{AB} = (A, \tilde{B})$. We have the weight function $w_b : \tilde{B} \to \mathbb{N}$ such that for all $v \in \tilde{B}, w_b(v) = |h^{-1}(v)| \leq \ell$, as the size of components in $G[V \setminus S]$ is at most ℓ. We have that $\sum_{v \in \tilde{B}} w_b(v) \geq (2\ell-1)|A|$ and there are no isolated vertices in $\tilde{B}$. Hence, (A, B) is the desired reducible pair. □

Lemma 7.17. *Let (X, Y) be a reducible pair. Then, there exists a partition of $X \cup Y$ into $C_1, ..., C_{|X|}$ such that (i) for all $u_i \in X$, we have $u_i \in C_j$ if and only if $i = j$, (ii) for all $i \in [|X|]$, $|C_i| \geq \ell + 1$, (iii) for every component C in $G[Y]$, there exists a unique C_i such that $V(C) \subseteq C_i$ and $u_i \in N(C)$ and (iv) if (X, Y) is a strict reducible pair, then there exists C_j such that $|C_j| \geq 2\ell + 1$.*

Proof. Construct $\tilde{G}_{XY} := (X, \tilde{Y})$. Run the algorithm of Lemma 7.10 with input $\tilde{G}_{XY}, q = 2\ell - 1$, and $W = \ell$(as the capacity of any vertex in $\tilde{Y}$ is at most ℓ) which outputs an unsplitting weighted ℓ-expansion f' in $\tilde{G}_{XY}$. In polynomial time, we modify f' such that if there is a vertex $v \in \tilde{Y}$ such that $\forall u \in N(v), f'(uv) = 0$, we choose a vertex $u \in N(v)$ and set $f'(uv) = w_b(v)$. For each $u_i \in X$ define $C_i := u_i \bigcup_{f'(u_i v) \neq 0} h^{-1}(v)$. Since f' is unsplitting, the collection $C_1, \ldots, C_{|X|}$ forms a partition of $X \cup Y$. By the definition of C_i, we have that for any $u_i \in X$, $u_i \in C_j$ if and only if $i = j$. For any component C in $G[Y]$, $h(C)$ is matched to a unique vertex $u_i \in X$ by f', we have that $V(C) \subseteq C_i$. As f' is a weighted ℓ-expansion, $|C_i| = 1 + \sum_{f'(u_i v) \neq 0} |h^{-1}(v)| = 1 + \sum_{f'(u_i v) \neq 0} f'(u_i v) \geq 1 + \ell$. Let (X, Y) be strict at $u_j \in X$. Then, we can use Lemma 7.10 to obtain the expansion f' such that it is strict at u_j. Hence, $|C_j| = 1 + \sum_{f'(u_j v) \neq 0} |h^{-1}(v)| = 1 + \sum_{f'(u_j v) \neq 0} f'(u_j v) > 1 + \ell + (\ell - 1)$ which implies $|C_j| \geq 2\ell + 1$. This concludes the proof of the lemma. □

Lemma 7.18. *Let (X, Y) be a reducible pair. If (G, k) is a* YES*-instance for ℓ-COC, then there exists an ℓ-COC solution S of size at most k such that $X \subseteq S$ and $S \cap Y = \emptyset$.*

Proof. By Lemma 7.17 we have that there are $C_1, \ldots, C_{|X|} \subseteq X \cup Y$ vertex disjoint sets of size at least $\ell + 1$ such that for all $i \in [|X|]$, $G[C_i]$ is a connected set. Let S' be an arbitrary solution. Then, S' must contain at least one vertex from each C_i. Let $S := S' \setminus (X \cup Y) \cup X$. We have that $|S| \leq |S'| - |X| + |X| = |S'|$. As any connected set of size $\ell + 1$ that contains a vertex in Y also contains a vertex in X and $X \subseteq S$, S is also an ℓ-COC solution. □

Now we encode an ℓ-COC instance (G, k) as an INTEGER LINEAR PROGRAMMING instance. We introduce $n = |V(G)|$ variables, one variable x_v for each vertex $v \in V(G)$. Setting the variable x_v to 1 means that v is in S, while setting $x_v = 0$ means that v is not in S. To ensure that S contains a vertex from every connected set of size $\ell + 1$, we can introduce constraints $\sum_{v \in C} x_v \geq 1$ where C is a connected set of size $\ell + 1$. The size of S is given by $\sum_{v \in V(G)} x_v$. This gives us the following ILP formulation:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V(G)} x_v, & \\ \text{subject to} & \sum_{v \in C} x_v \geq 1 & \text{for every connected set } C \text{ of size } \ell + 1 \\ & 0 \leq x_v \leq 1 & \text{for every } v \in V(G) \\ & x_v \in \mathbb{Z} & \text{for every } v \in V(G). \end{array}$$

Note that there are $n^{\mathcal{O}(\ell)}$ connected sets of size at most ℓ in a graph on n vertices. Hence, providing an explicit ILP requires $n^{\mathcal{O}(\ell)}$ time which forms the bottleneck for the running time of the kernelization algorithm that follows. We consider the Linear Programming relaxation of above ILP obtained by dropping the constraint that $x \in \mathbb{Z}$. By an optimal LP solution S_L with weight L we mean the set of values assigned to each variable, and optimal value is L. For a set of vertices $X \in V(G)$, $X = 1$ ($X = 0$) denotes that every variable corresponding to vertices in X is set to 1 (0).

Lemma 7.19. *Let S_L be an optimal LP solution for G such that $x_v = 1$ for some $v \subseteq V(G)$. Then, $S_L - x_v$ is an optimal LP solution for $G - v$ of value $L - 1$.*

Proof. Clearly, $S_L - x_v$ is feasible solution for $G - v$ of value $L - 1$. Suppose it is not optimal. Let $S_{L'}$ be an optimal LP solution for $G - v$ such that $L' < L - 1$. Then, $S_{L'} \cup x_v$ with $x_v = 1$ is an optimal LP solution for G with value $< L - 1 + 1 = L$ contradicting that the optimal solution value of LP for G is L. □

From now on by running LP after setting $x_v = 1$ for some vertex v, we mean running the LP algorithm for $G - v$ and including $x_v = 1$ in the obtained solution to get a solution for G.

Lemma 7.20. *Let (X, Y) be a strict reducible pair. Then every optimal LP solution sets at least one variable corresponding to a vertex in X to 1.*

Proof. By Lemma 7.18, we have that every connected set of size $\ell + 1$ in $G[X \cup Y]$ contains a vertex in X. Hence, from any LP solution S_L, a feasible LP solution can be obtained by setting $X = 1$ and $Y = 0$. Since, we have at least $|X|$ many vertex disjoint LP constraints, for each $v_i \in X$, we have $\sum_{u \in C_i} x_u = 1$. By Lemma 7.17, there is a set $C_j \subseteq X \cup Y$ such that $|C_j| \geq 2\ell + 1$. If $x_{v_j} \neq 1$, then there is a vertex $w \in C_j$ such that $x_w > 0$. Let $w \in C \subset C_j$ where $G[C]$ is a connected component in $G[Y]$. Since $|C| \leq \ell$, there is a connected set C' of size at least $\ell + 1$ in $G[C_j] - C$. But now $\sum_{u \in C'} x_u < 1$ contradicting that S_L is feasible. □

Lemma 7.21. *Let (X, Y) be a minimal reducible pair. If for any vertex $v \in X$, an optimal LP solution sets $x_v = 1$, then it also sets $X = 1$ and $Y = 0$.*

Proof. We prove the lemma by contradiction. Let $X' \subset X$ be the largest subset of X such that $X' = 1$. Consider $\tilde{G}_{XY}$. Let $Y' \subseteq \tilde{Y}$ be the set of vertices such that $N(Y') \subseteq X'$. Let $Z := \bigcup_{v \in Y'} h^{-1}(v)$. By the minimality of (X, Y), we have that $\sum_{v \in Y'} w(v) < (2\ell - 1)|X'|$. Hence, $\sum_{v \in \tilde{Y} \setminus Y'} w(v) > (2\ell - 1)|X \setminus X'|$. Clearly, the weighted $(2\ell - 1)$-expansion in the reducible pair (X, Y) when restricted to $(X \setminus X', Y \setminus Z)$ provides a weighted $(2\ell - 1)$-expansion of $X \setminus X'$ into $Y \setminus Z$. This implies that $(X \setminus X', Y \setminus Z)$ is a strict reducible pair in $G - (X' \cup Z)$. By Lemma 7.19, we have that the LP solution restricted to $G - (X' \cup Z)$ is optimal. Since $(X \setminus X', Y \setminus Z)$ is a strict reducible pair, by Lemma 7.20, there is a vertex $u \in X \setminus X'$ such that $x_u = 1$, but this contradicts the maximality of X'. Therefore, if for any vertex $v \in X$, an LP solution sets $x_v = 1$, then it sets $X = 1$ and $Y = 0$. □

Lemma 7.22. *There exists a polynomial time algorithm that given an integer ℓ and ℓ-COC instance (G, k) on at least $2\ell k$ vertices either finds a reducible pair (X, Y) or concludes that (G, k) is a* NO*-instance.*

Proof. If (G, k) is a YES-instance of ℓ-COC, then by Lemma 7.16, there exists a reducible pair (X, Y). We use the following algorithm to find one:

Step 1 Run the LP algorithm. Let $A = 1$ and $B = 0$ in the LP solution.

Step 2 If both A and B are non-empty, then run the algorithm of Lemma 7.15 with input $(G, k), A, B$. If it outputs a reducible pair (X, Y), then return (X, Y) and terminate. Otherwise, go to step 3.

Step 3 Now we do a linear search for a vertex in X. For each vertex $v \in V(G)$, do the following: in the original LP introduce an additional constraint that sets the value of the variable x_v to 1 i.e. $x_v = 1$ and run the LP algorithm. If the optimal value of the new LP is the same as the optimal value of the original LP, then let $A = 1$ and $B = 0$ be the sets of variables set to 1 and 0 respectively in the optimal solution of the new LP and go to step 2.

Step 4 Output a trivial NO-instance.

Step 1 identifies the set of variables set to 1 and 0 by the LP algorithm. By Lemma 7.21, we have that if there is a minimal reducible pair (X, Y) in G, then $X \subseteq A$ and $Y \subseteq B$. So, in Step 2 if the algorithm succeeds in finding one, we return the reducible pair and terminate otherwise we look for a potential vertex in X and set it to 1. If (X, Y) exists, then for at least one vertex, setting $x_v = 1$ would set $X = 1$ and $Y = 0$ (by Lemma 7.21) without changing the LP value and we go to Step 2 to find it. If for each choice of $v \in V(G)$, the LP value changes when x_v is set to 1, we can conclude that there is no reducible pair and output a trivial NO instance. Since, we need to do this search at most n times and each step takes only polynomial time, the total time taken by the algorithm is polynomial in the input size. □

Theorem 21. *For every constant $\ell \in \mathbb{N}$, ℓ-Component Order Connectivity admits a kernel with at most $2\ell k$ vertices that takes $n^{\mathcal{O}(\ell)}$ time.*

7.5 Separation oracle for ℓ-COC

We resume our discussion of separation oracles from Section 4.7. For ℓ-COC, we have an LP with $n^{\ell+1}$ constraints: for every connected set of size $\ell + 1$, we need the sum of the variables to be at least 1. So a separation oracle for this LP should take an input G and variable values x_v on all the vertices and then determine whether there exists a connected set of $\ell + 1$ vertices with sum less than 1. If yes, then that is a violated constraint. Let us first define this problem formally:

Min ℓ-Connected-Subgraph (ℓ-MCS)

Input:	A graph G and vertex weights $w : V \to \mathbb{R}$.
Question:	Find a connected subgraph C on at least $\ell + 1$ vertices such that the weight of subgraph C is less than 1, if one exists where weight of a subgraph is the sum of weights of vertices in it.

Now, we prove that ℓ-MCS is NP-complete by reducing from Set Cover.

Theorem 22. MIN ℓ-CONNECTED-SUBGRAPH *is* NP*-complete.*

Proof. Let $I := (U, \mathcal{H})$ be an instance of SET COVER such that $|U| = n$ and $|\mathcal{H}| = m$. Construct the set-element incidence (bipartite) graph $G := (A, B, E)$ for I as follows: A contains a vertex for each set in $\mathcal{H}$ and B contains a vertex for each element in U. Two vertices $u \in A, v \in B$ are adjacent if and only if the element corresponding to v belongs to the set corresponding to u. In addition, B contains a special vertex b which is adjacent to every vertex in A. We define the weight function as follows: $w : V(G) \to \mathbb{R}$ such that $\forall u \in A, w(u) = \frac{1}{k+1}$ and $\forall v \in B, w(v) = 0$. Finally, we set $\ell := n + k$.

We claim that I has a set cover of size k if and only if G has a connected subgraph C of size $\ell + 1$ and weight less than 1. For the forward direction, assume $\mathcal{F}$ is a set cover of size k. Let $H \subseteq A$ corresponding to elements in $\mathcal{F}$. Then, $G[H \cup B]$ is a connected set on $k + n + 1$ vertices: every vertex in $B - b$ has an edge to at least one vertex in H and b is adjacent to every vertex in H.

For the backward direction, let C be a connected graph on $n + k + 1$ vertices and weight less than 1. Clearly, $|A \cap V(C)| < k + 1$, otherwise weight of C would be at least 1. Rest of the vertices must belong to B. As C is connected and B is independent, every vertex in B must have an edge in $A \cap V(C)$. Hence, the set corresponding to elements in $A \cap V(C)$ form a set cover of size k. □

To find a connected subgraph of minimum weight, we use color coding. The technique of color coding was introduced by Alon, Yuster and Zwick [AYZ95]. Suppose the size (number of vertices) in the sought subgraph H be k. If we color the vertex set of G, an n-vertex graph, using k colors where each vertex is assigned one of the k colors uniformly and independently at random, then with probability at least e^{-k}, the vertices of H are colored with pairwise distinct colors. To justify this argument it is easy to see that there are k^n possible colorings of $V(G)$ and $k!k^{n-k}$ of these colorings are such that $V(H)$ has pairwise distinct colors which implies the probability of success to be at least $\frac{k^n}{k!k^{n-k}} \geq e^{-k}$. A typical sequence of steps is to first obtain a random coloring of G, then find a *colorful* subgraph H in G. To improve the success probability, repeat the above steps $O(e^k)$ times.

Now we describe the oracle implementation formally. Given a graph G and a coloring $c : V(G) \to [k]$, a subgraph H of G is called *colorful* if the vertices in $V(H)$ get pairwise distinct colors under c. Note that c need not be a proper coloring of G in which we require endpoints of an edge to have distinct colors. In our case, H is a connected subgraph on $\ell + 1$ vertices. As the first step of the algorithm, we color the vertices in $V(G)$ uniformly and independently at random. As argued above, with probability at least $e^{-(\ell+1)}$, a connected set H with minimum total weight gets multicolored. Now our task is to find a minimum weight multicolored connected set for which we use dynamic programming.

Theorem 23. *Let G be an undirected, weighted graph with weights $w : V(G) \to \mathbb{R}$ and let $c : V(G) \to [k]$ be a coloring of its vertices with k colors. There exists a deterministic algorithm that checks in time $3^k n^{\mathcal{O}(1)}$ whether G contains a colorful connected subgraph on k vertices and, if this is the case, returns one such subgraph of smallest weight.*

Proof. We assume that G is a connected graph. If G is not a connected graph, then the algorithm described below is run on each connected component, and the output of the algorithm is the logical OR of outputs corresponding to each component.

Define a dynamic programming table T that takes as input a subset of colors S and a vertex v and returns the minimum weight of a multicolored connected subset that uses v and all the colors in S exactly once. If v is not colored with a color in S, then set $T[S, v] = \infty$. Otherwise,

$$T[S, v] = \min_{S' \subseteq S, v' \in N(v)} \{T[S \setminus S', v] + T[S', v']\} \tag{7.1}$$

In Equation 7.1, the subset S' is such that the color of v is not in S' and v' is a neighbor of v. Finally, the algorithm returns the smallest value in the last row of the table. If this value is ∞, then G does not contain any colorful connected set on k vertices. Otherwise, one can recover the multicolored connected set via back-tracking.

To argue about the correctness of the algorithm, it is sufficient to show the correctness of the Equation 7.1. We need to show two inequalities. First, we show that $T[S, v] \leq \min_{S' \subseteq S, v' \in N(v)}\{T[S \setminus S', v] + T[S', v']\}$. Observe that for any $v' \in N(v)$, the union of connected sets corresponding to $T[S \setminus S', v]$ and $T[S', v']$ is also a connected set and as $S \setminus S'$ and S' are disjoint, the union is colorful as well. Hence, $T[S, v] \leq T[S \setminus S', v] + T[S', v']$.

For the other inequality, let H be a connected set corresponding to $T[S, v]$. There is a partition (A, B) of H such that $v \in A$, B contains a neighbor of v and both $G[A]$ and $G[B]$ are connected sets:

Remove v from H. If $H \setminus v$ is connected let $A := v$ and $B := H \setminus v$. Otherwise, let A be defined as the union of v with all connected components of $H \setminus v$ except one (say, set C) and B has the component C.

Let S' be the set of colors of vertices in B and $S \setminus S'$ be the color of vertices in A. Note that $v \in A$. Let v' be a neighbor of v in B. By definition, $T[S \setminus S', v] \leq w(A)$ and $T[S', v'] \leq w(B)$ where $w(A)$ and $w(B)$ are the sum of weights of vertices in A and B respectively. Hence, $T[S, v] = w(A) + w(B) \geq T[S \setminus S', v] + T[S', v']$. This concludes the proof of correctness of the algorithm.

Observe that there are $2^{|S|} \cdot n$ terms in the equation 7.1. Hence, each of entries $T[S, v]$ can be computed in $2^{|S|}n^{\mathcal{O}(1)}$ time. There are $\binom{k}{|S|}$ many subsets of colors of size $|S|$. Hence, the running time of the algorithm is bounded by $\sum_{i=0}^{k} \binom{k}{i} 2^i n^{\mathcal{O}(1)} = 3^k n^{\mathcal{O}(1)}$. □

Derandomization. Algorithms based on color coding are randomized, but one can often derandomize these algorithms. The basic idea of derandomization is as follows: instead of picking a random coloring $c : [n] \to [k]$, we deterministically construct a family $\mathcal{F}$ of functions $f : [n] \to [k]$ such that it is guaranteed that one of the functions from $\mathcal{F}$ has the property that we hope to attain by choosing a random coloring c.

Definition 7.23. An (n, k, ℓ)-splitter $\mathcal{F}$ is a family of functions from $[n]$ to $[\ell]$ such that for every set $S \subseteq [n]$ of size k there exists a function $f \in \mathcal{F}$ that splits S evenly. That is, for every $1 \leq j, j' \leq \ell$, $|f^{-1}(j) \cap S|$ and $|f^{-1}(j') \cap S|$ differ by at most 1.

Theorem 24 ([AYZ95])**.** *For any $n, k \geq 1$ one can construct an (n, k, k^2)-splitter of size $k^{\mathcal{O}(1)} \log n$ in time $k^{\mathcal{O}(1)} n \log n$.*

Definition 7.24. An (n, k, k)-splitter is called an (n, k)-perfect hash family.

Theorem 25 ([NSS95])**.** *For any $n, k \geq 1$ one can construct an (n, k)-perfect hash family of size $e^k k^{\mathcal{O}(\log k)} \log n$ in time $e^k k^{\mathcal{O}(\log k)} n \log n$.*

Let (G, k) be the input instance for ℓ-COC, where $n = |V(G)|$. Instead of taking a random coloring c of $V(G)$, we use Theorem 25 to construct an (n, k)-perfect hash family $\mathcal{F}$. Then, for each $f \in \mathcal{F}$, we invoke the dynamic programming algorithm of Theorem 23 for the coloring $c := f$. The properties of an (n, k)-perfect hash family $\mathcal{F}$ ensure that, if there exists a connected set H on $\ell + 1$ vertices in G, there there exists $f \in \mathcal{F}$ that is injective on $V(H)$ and, consequently, the algorithm of Theorem 23 finds a colorful subgraph H for the coloring $c := f$. As a consequence, we have the following improvement in Theorem 21.

Theorem 26. *For every constant $\ell \in \mathbb{N}$, ℓ-Component Order Connectivity admits a kernel with at most $2\ell k$ vertices that can be computed in $(3e)^\ell \cdot n^{\mathcal{O}(1)}$ time.*

Chapter 8

Connected Dominating Set

8.1 Beyond kernelization

As many real-world computational problems are NP-hard, the first question an algorithm designer should answer when faced with large instances of such problems is whether polynomial-time preprocessing can be applied. Ideally, these preprocessing routines should run very fast (at least faster than the algorithms for solving the problem) and little to no loss of information about solutions should occur. In the past decade or so, progress in parameterized complexity [DF97] provided a robust mathematical framework to analyze the performance of preprocessing routines. In parameterized complexity, we consider instances (I, k) of a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. Typically, I is an instance of some computational problem, and k denotes the parameter which reflects some structural property of the instance. The *natural parameter* is a bound on the size of an optimal solution. A parameterized problem is said to admit a *kernel* if there is a polynomial-time algorithm, called a *kernelization* algorithm, that reduces the input instance down to an instance whose size is bounded by a function $f(k)$ in k, while preserving the answer. If $f(k)$ is a linear, polynomial, or exponential function of k, we say that this is a linear, polynomial, or exponential kernel respectively. Over the last decade, kernelization has become a very active field of study, especially with the development of complexity-theoretic tools to show that a problem is not likely to admit a polynomial kernel [BDFH09, Dru15, FS11], or a kernel of a specific size [DM12, DvM14b, HW12].

Unfortunately, going back to our real-world problem instances, it turns out that the notion of polynomial kernels has a few limitations. In addition to the fact that many typical problems have been shown not likely to admit polynomial kernels, the formal definition of kernelization does not play well with approximation algorithms. In other words, if our goal is not to solve an instance optimally (as would be expected for most NP-hard problems as even the most efficient algorithms for solving them would take exponential time) then, for a large number of problems, it is not possible to translate an approximate solution to the instance (I', k') into an approximate solution to the original instance (I, k). Indeed, given anything but an optimal solution to (I', k'), it is (for many problems) impossible to draw

conclusion about the original instance. To address these issues, Lokshtanov et al. [LPRS16] developed the framework of *lossy kernelization*. Intuitively, the framework combines notions from approximation and kernelization algorithms to allow for approximation preserving kernels. In this framework, for $\alpha > 1$, an *α-approximate kernel* is a polynomial-time algorithm that takes as input an instance (I, k) and outputs an instance (I', k') of the same problem such that size of (I', k') is upper bounded by $f(k)$, for a function f independent of $|I|$, and for every $c \geq 1$, a c-approximate solution for the new instance can be turned into a $c\alpha$-approximate solution of the original instance in polynomial time. In the case of an *α-approximate bikernel*, the reduced instance need not be an instance of the same problem.

Connected Domination Set

Both Dominating Set and Connected Dominating Set are NP-hard even on very restricted graph classes, e.g. planar graphs of bounded degree [GJ79a]. Naturally, the complexity of the problems has been extensively studied under different algorithmic frameworks, notably approximation algorithms and parameterized complexity. In parameterized complexity, Dominating Set (and hence Connected Dominating Set) is a standard example of a W[2]-complete problem. This is widely believed to exclude the possibility of having FPT algorithms, thereby no kernels whatsoever. Therefore, attempts have been made to find graph classes on which the results are not so discouraging. This line of research has led to very fruitful and insightful discoveries in parameterized complexity in general and in kernelization in particular. For the Connected Dominating Set problem, which admits an $\mathcal{O}(\log n)$-approximation algorithm [GK98], linear kernels are only known for planar [LMS11] and H-topological-minor-free graphs [FLST13]. Polynomial kernels are excluded already for graphs of bounded degeneracy [CPPW12] and for graphs of bounded expansion [DDF+16]. In this chapter we prove the following theorem.

Theorem 27. *For every $\epsilon > 0$,* Connected Dominating Set*, parameterized by solution size, admits a $(1+\epsilon)$-approximate kernel with $k^{\mathcal{O}(\frac{d^2}{\epsilon})}$ vertices on $K_{d,d}$-free graphs and a $(1+\epsilon)$-approximate bikernel with $\mathcal{O}(g(\epsilon)k)$ vertices on graphs of bounded expansion, where $g(\epsilon)$ is a constant depending only on ϵ.*

Unfortunately, there is a minor technical detail (which also can be found in [DDF+16] for the case of r-Dominating Set) that prevents us from obtaining a reduced instance of the same problem when dealing with graphs of bounded expansion. Instead, we kernelize to an annotated version of the problem, where only a given subset of vertices of G needs to be dominated by a connected set. Although the technical details for dealing with the two graph classes are quite different, the high-level approach is identical. Both kernelization algorithms follow the same two-step strategy. First, our goal is to compute a "small" set of vertices whose domination is sufficient, i.e. the set of *dominatees* or the so-called domination core. The exact definition of a domination core will be different for the two graph

classes but intuitively it is simply a set of vertices whose domination guarantees the domination of the whole graph. Having found a domination core of appropriate size, the next step is to reduce the number of *dominators*, i.e. vertices whose role is to dominate other vertices, and the number of *connectors*, i.e. vertices whose role is to connect the solution. When reducing the number of vertices outside the domination core, we borrow approximation techniques that are closely related to the Steiner Tree problem. In the Steiner Tree problem, we are given as input a graph G and a set $T \subseteq V(G)$ of terminals. The objective is to find a subtree of G spanning T that minimizes the number of vertices. Intuitively, we find it easy to think about our approach as follows. The first step borrows ideas from kernelization algorithms for Dominating Set and the second step borrows ideas from approximation algorithms for the Steiner Tree problem. By carefully combining the two and proving additional combinatorial results, we obtain the claimed lossy kernels. Note that the size of our kernel for graphs of bounded expansion matches (degree of the polynomial) the size of the best known kernel for Dominating Set, while for $K_{d,d}$-free graphs we have an additional $\frac{1}{\epsilon}$ multiplicative factor in the exponent.

8.2 Preliminaries

8.2.1 Basics of graphs, sparse graphs, and notations

We use standard terminology from the book of Diestel [Die12] for those graph-related terms that are not explicitly defined here. All graphs we consider are finite, simple, and undirected. For a graph G, $V(G)$ and $E(G)$ denote the vertex and edge sets of the graph, respectively. We let $|G| = |V(G)|$ denote the number of vertices in G and $||G|| = |E(G)|$ denote the number of edges in G. The *density* of a graph G, density(G), is defined as density$(G) = ||G||/|G|$. The *radius* of G, denoted radius(G), is the minimum integer r such that there exists a vertex $v \in V(G)$, called the *center*, which is at distance at most r from all vertices in $V(G)$, i.e. the shortest path between v and any other vertex of G has at most $r-1$ internal vertices. For a vertex $v \in V(G)$, we let $N_G(v) = \{u \mid uv \in E(G)\}$ denote the *open neighborhood* of v in G. The *closed neighborhood* of v is denoted by $N_G[v] = N_G(v) \cup \{v\}$. For a set $X \subseteq V(G)$, we let $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. When the graph is clear from context we omit the subscript G. We sometimes use the notation $N_X(v) = N_G(v) \cap X$ and refer to it as the *X-neighborhood* of v. The *degree* of v, $\deg(v)$, is the number of neighbors of v, i.e. $\deg(v) = |N_G(v)|$. For a positive integer r and a vertex $v \in V(G)$, we let $N_G^r(v)$ denote a ball of radius r around v, i.e. the set of all vertices in G that are at distance at most r from v. For a vertex subset $X \subseteq V(G)$, $G[X]$ and $G - X$ are the graphs induced on X and $V(G) \setminus X$, respectively. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G[V(H)])$.

Given a graph G and two vertex subsets $D, Z \subseteq V(G)$, we say that D is a *Z-dominator* if D dominates Z in G, i.e. every vertex $z \in Z \setminus D$ is at distance

at most one from some vertex in D. A vertex v dominates itself and all its neighbors. We denote by $\mathbf{ds}(G, Z)$ ($\mathbf{cds}(G, Z)$) the size of a smallest (connected) Z-dominator in G. By $\mathbf{ds}(G)$ ($\mathbf{cds}(G)$) we mean $\mathbf{ds}(G, V(G))$ ($\mathbf{cds}(G, V(G))$). For a set $T \subseteq V(G)$, we denote by $\mathbf{st}_G(T)$ the size of, i.e. the number of vertices in, the smallest Steiner tree connecting T in G (including vertices in T).

Proposition 8.1. *Let G be a graph, $Z \subseteq V(G)$, such that $G[Z]$ is connected, and let D be a Z-dominator such that $G[D]$ has at most p connected components. Then a set $Q \subseteq Z$ of size at most $2p$ such that $G[D \cup Q]$ is connected, can be computed in polynomial time.*

Proof. If D is connected, then we set $Q = \emptyset$ and the proposition follows. Hence, assume that $G[D]$ contains at least 2 connected components. Now, the algorithm for finding Q does the following:
(1) We start by setting $Q = \emptyset$.
(2) If $G[D \cup Q]$ is connected, output Q.
(3) If there is a vertex z in $Z \setminus (D \cup Q)$ that is dominated from at least 2 different components of $G[D \cup Q]$, we add z into Q.
(4) If there is an edge uv with both end vertices in Z such that the number of components of $G[D \cup Q]$ that dominate at least one of $\{u, v\}$ is at least 2, we add u and v into Q.

Note that in both steps (3) and (4) we increase the size of Q by at most two and decrease the number of connected components in $G[D \cup Q]$ by at least one. Hence after at most $p - 1$ steps we end up with only one connected component. Now we show that if we cannot apply step (3) nor step (4), then $G[D \cup Q]$ is connected. Since, we cannot apply step (3), each vertex of Z is dominated by exactly one connected component of $D \cup Q$. Moreover, since we cannot apply step (4), if uv is an edge in Z, then both u and v are dominated by the same component of $G[D \cup Q]$. Finally, since Z is connected, there is a path between every pair of vertices of Z and all vertices on that path have to be dominated by the same distinct component of $G[D \cup Q]$, which implies that $G[D \cup Q]$ consists of a single connected component. □

Definition 8.2. Let D be a connected graph and $t \in \mathbb{N} \setminus \{0\}$. A (D, t)-*covering family* is a family $\mathcal{F}(D, t)$ of connected subgraphs of D such that (i) for each $T \in \mathcal{F}(D, t)$, $|V(T)| \leq 2t$ and (ii) $\bigcup_{T \in \mathcal{F}(D,t)} V(T) = V(D)$.

Proposition 8.3. *Let D be a connected graph and $t \in \mathbb{N} \setminus \{0\}$. Then there is a (D, t)-covering family $\mathcal{F}(D, t)$ such that $|\mathcal{F}(D, t)| \leq \frac{|V(D)|}{t} + 1$, and $\sum_{T \in \mathcal{F}(D,t)} |V(T)| \leq (1 + \frac{1}{t})|V(D)| + 1$.*

Proof. Let T_D be a spanning tree of D. We create a (D, t)-covering family $\mathcal{F}(D, t) = \{T_1, T_2, \ldots\}$, which is a set of subtrees of T_D constructed as follows. We root T_D at an arbitrary vertex $r \in V(T_D)$. For any pair of vertices $u, v \in V(T_D)$, u is called a child of v if $uv \in E(T_D)$ and v lies on the path from u to r. For each vertex $v \in V(T_D)$, we let $\text{weight}(v) = 1 + \sum_{u \in \text{child}(v)} \text{weight}(u)$, where $\text{child}(v)$ denotes the set of children of v in T_D. In other words, $\text{weight}(v)$ is the number of

vertices in the subtree rooted at v. Leaves have weight one. We use T_v to denote the subtree rooted at v. We construct $\mathcal{F}(D,t) = \{T_1, T_2, \ldots\}$ from T_D as follows:

1. If T_D is empty, terminate. Otherwise, compute the weights of all vertices in T_D then sort the vertices in increasing order of weight.

2. If there exists a vertex whose weight is between t and $2t$ (inclusive), pick the vertex with the smallest such weight, add T_v to $\mathcal{F}(D,t)$, delete T_v from T_D, then go back to step (1).

3. If there exists a vertex whose weight is strictly greater than $2t$, pick the vertex with the smallest such weight, greedily compute a subset $S \subseteq \text{child}(v)$ such that $t < \sum_{u \in S} \text{weight}(u) < 2t$, let $R = \text{child}(v) \setminus S$, add $T_v - \bigcup_{w \in R} V(T_w)$ to $\mathcal{F}(D,t)$, delete T_u from T_D, for every $u \in S$, then go back to step (1). Note that by our choice of v, all children of v must have weight at most $t-1$ as otherwise case (2) would apply.

4. Otherwise, every vertex in T_D has weight strictly less than t and hence T_D has at most t vertices (by the definition of the weight function). In this case, simply add T_D to $\mathcal{F}(D,t)$ and terminate.

It is not hard to see that whenever we delete a subtree in the above procedure we never disconnect the tree. Moreover, the procedure terminates only when the tree becomes empty and hence $\bigcup_{T \in \mathcal{F}(D,t)} V(T) = V(T_D)$. Whenever a tree is added to $\mathcal{F}(D,t)$ in either step (2) or (3), the size of that tree is at least equal to t and at most equal to $2t$. For step (4), the size of the added tree is strictly less than t. Combining those two facts, we know that $|\mathcal{F}(D,t)|$ is at most $\frac{|V(D)|}{t} + 1$, as the size of the current tree T_D is reduced by at least t for all but one subtrees. To prove the last inequality, i.e. $\sum_{T \in \mathcal{F}(D,t)} |V(T)| \leq (1 + \frac{1}{t})|V(D)| + 1$, we let $\text{mult}(v)$ denote the multiplicity of v minus one, i.e. the number of subtrees $T \in \mathcal{F}(D,t)$ in which v appears minus one. So if a vertex occurs only once, its multiplicity is zero. Observe that $\sum_{T \in \mathcal{F}(D,t)} |V(T)| \leq |V(D)| + \sum_{v \in V(T_D)} \text{mult}(v)$. However, whenever the multiplicity of a vertex increases by one, the size of the running subtree decreases by at least t (step (3)) or the procedure terminates (step (4)). Hence, $\sum_{v \in T_D} \text{mult}(v) \leq \frac{|V(D)|}{t} + 1$, and the bound follows. □

A set $S \subseteq V(G)$ is *ℓ-scattered* in G if for every pair of distinct vertices in S the distance between them is at least $\ell + 1$ in G. A useful observation which follows from this definition is that if there exists a 2-scattered set S of size k in G, then every dominating set of G must have size at least k since every vertex of G can dominate at most one vertex of S. A *clique* in a graph is a subset of pairwise adjacent vertices. An *independent set* is a subset of pairwise non-adjacent vertices. We let K_c and I_c denote a clique on c vertices and an independent set on c vertices, respectively. We let $K_{i,j}$ denote a biclique (a complete bipartite graph) with i vertices in one partition and j vertices in the other.

Sparse graphs We define the main classes we consider. We refer the reader to [BLS99, NdM10] for more details. *Contracting* an edge uv of G results in a new graph H in which the vertices u and v are deleted and replaced by a new vertex w that is adjacent to every vertex in $N_G(\{u, v\})$. If a graph H can be obtained from G by repeatedly contracting edges, H is said to be a *contraction* of G. If H is a subgraph of a contraction of G, then H is a *minor* of G.

Definition 8.4 (Shallow minors)**.** A graph M is an *r-shallow minor* of G, for some integer r, if there exists a family of disjoint subsets $V_1, \ldots, V_{|M|}$ of $V(G)$ such that:

- each graph $G[V_i]$ is connected and has radius at most r, and
- there is a bijection $\omega : V(M) \to \{V_1, \ldots, V_{|M|}\}$ such that for every edge $uv \in E(M)$ there is an edge in G with one endpoint in $\omega(u)$ and another in $\omega(v)$.

The set of all r-shallow minors of a graph G is denoted by $G\nabla r$. The set of all r-shallow minors of all members of a graph class $\mathcal{G}$ is denoted by $\mathcal{G}\nabla r = \bigcup_{G\in\mathcal{G}}(G\nabla r)$.

Definition 8.5 (Grad and bounded expansion)**.** For a graph G and an integer $r \geq 0$, the *greatest reduced average density (grad) at depth r* is, $\nabla_r(G) = \max_{M\in G\nabla r}\text{density}(M) = \max_{M\in G\nabla r}||M||/|M|$. For a graph class $\mathcal{G}$, $\nabla_r(\mathcal{G}) = \sup_{G\in\mathcal{G}}\nabla_r(G)$. A graph class $\mathcal{G}$ has *bounded expansion* if there is a function $f : \mathbb{N} \to \mathbb{R}$ such that for all r we have $\nabla_r(\mathcal{G}) \leq f(r)$.

For ease of presentation, and since we deal with several constants, we shall use the following convention. We assume that a graph class of bounded expansion $\mathcal{G}$ is fixed, and hence so are the values of $\nabla_i(\mathcal{G})$, for all non-negative integers i. This assumption is not strictly required but it significantly simplifies the analysis.

Recall that a class of graphs $\mathcal{G}$ has *bounded degeneracy* if every induced subgraph of a graph $G \in \mathcal{G}$ has a vertex of degree at most d, for some constant d. Note that bounded expansion implies bounded degeneracy, since the degeneracy of a graph G is equal to $2\nabla_0(G)$ (as $G\nabla 0$ contains exactly the subgraphs of G). Moreover a d-degenerate graph cannot contain $K_{d+1,d+1}$ as a subgraph, which brings us to the class of biclique-free graphs.

Definition 8.6. A class of graphs $\mathcal{G}$ is said to be *d-biclique-free*, for some $d > 0$, if $K_{d,d}$ is not a subgraph of any $G \in \mathcal{G}$, it is said to be *biclique-free* if it is d-biclique-free for some d.

Again, we assume a fixed biclique-free graph class, hence a fixed value of d. All of our arguments can be easily extended to $K_{i,j}$-free graphs, for non-negative integers i and j, but we use $K_{d,d}$-freeness for simplicity. Next, we state some useful properties of graphs of bounded expansion and biclique-free graphs that will be used later on.

Lemma 8.7 ([DDF+16]). *Let G be a graph and let G' be the graph obtained by adding a universal vertex to G, i.e. a vertex adjacent to all vertices in G. Then, for any non-negative integer r, $\nabla_r(G') \leq \nabla_r(G) + 1$.*

Given two graphs G and H, the *lexicographic product* $G \odot H$ is defined as the graph on the vertex set $V(G) \times V(H)$ where vertices (u, a) and (v, b) are adjacent if $uv \in E(G)$ or if $u = v$ and $ab \in E(H)$.

Lemma 8.8 ([DDF+16]). *For a graph G and non-negative integers $t \geq 1$ and r we have $\nabla_r(G \odot K_t) \leq 4(8t(r+t) \cdot \nabla_r(G) + 4t)^{(r+1)^2}$.*

Let G be a graph and X be a subset of its vertices. For $u \in V(G) \setminus X$, we define the *r-projection* of u onto X as follows: $M_r^G(u, X)$ is the set of all vertices $w \in X$ for which there exists a path in G that starts in u, ends in w, has length at most r, and whose internal vertices do not belong to X. Note that $M_1^G(u, X) = N_X(u)$. We omit the superscript when the graph is clear from context.

Lemma 8.9 ([DDF+16]). *Let $\mathcal{G}$ be a class of graphs of bounded expansion. There exists a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, $X \subseteq V(G)$, and an integer $r \geq 1$, computes the* r-closure *of X, denoted by $cl_r(X)$, with the following properties: (i) $X \subseteq cl_r(X) \subseteq V(G)$, (ii) $|cl_r(X)| \leq C_{\mathbf{cl1}} \cdot |X|$, and (iii) $|M_r^G(u, cl_r(X))| \leq C_{\mathbf{cl2}}$ for each $u \in V(G) \setminus cl_r(X)$, where $C_{\mathbf{cl1}}$ and $C_{\mathbf{cl2}}$ are constants depending only on r and a fixed (finite) number of grads of $\mathcal{G}$.*

Lemma 8.10 ([DDF+16]). *Let $\mathcal{G}$ be a class of graphs of bounded expansion and $r \in \mathbb{N}$. Let $G \in \mathcal{G}$ be a graph and let $X \subseteq V(G)$. Then $|\{Y \mid Y = M_r(u, X)$ for $u \in V(G) \setminus X\}| \leq C_{\mathbf{ex}} \cdot |X|$, where $C_{\mathbf{ex}}$ is a constant depending only on r and a fixed (finite) number of grads of $\mathcal{G}$.*

Lemma 8.11 ([DDF+16]). *Let G be a bipartite graph with bipartition (X, Y) that belongs to some graph class $\mathcal{G}$ such that $\nabla_1(\mathcal{G}) \geq 1$. Moreover, suppose that for every $u \in Y$ we have that $N(u) \neq \emptyset$, and that for every distinct $u_1, u_2 \in Y$ we have $N(u_1) \neq N(u_2)$, i.e. Y is twin-free. Then there exists a mapping $\varphi : Y \to X$ with the following properties: (i) $u\varphi(u) \in E(G)$ for each $u \in Y$ and (ii) $\varphi^{-1}(v) \leq C_{\mathbf{ch}}$ for each $v \in X$, where $C_{\mathbf{ch}}$ is a constant depending only on a fixed (finite) number of grads of $\mathcal{G}$.*

Lemma 8.12 ([LMP+15]). *Let G be a $K_{d,d}$-free bipartite graph with bipartition (A, B) such that there are no two distinct vertices $u, v \in B$ with $N(u) = N(v)$. Then, $|B| \leq 2d|A|^d$.*

8.2.2 Lossy kernelization

Formally, we first require the notion of a *parameterized optimization problem*, which is the parameterized analogue of an optimization problem in the theory of approximation algorithms. Throughout this chapter we talk about only minimization problems and we refer to [LPRS16] to see the symmetric definitions related to maximization problems.

Definition 8.13. A *parameterized minimization problem* Π is a computable function $\Pi : \Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\infty\}$.

The instances of Π are the pairs $(I, k) \in \Sigma^* \times \mathbb{N}$ and a solution to (I, k) is simply a string $S \in \Sigma^*$ such that $|S| \leq |I| + k$. The value of a solution S is $\Pi(I, k, S)$. The optimum value of (I, k) is $\mathrm{OPT}_\Pi(I, k) = \min_{S \in \Sigma^*, |S| \leq |I|+k} \Pi(I, k, S)$ and we remove the subscript Π if it is clear from the context. An optimum solution for (I, k) is a solution S such that $\Pi(I, k, S) = \mathrm{OPT}_\Pi(I, k)$. Next we come to the notion of an *α-approximate polynomial-time preprocessing algorithm* for a parameterized minimization problem Π.

Definition 8.14. For $\alpha > 1$, an *α-approximate polynomial-time preprocessing algorithm* $\mathcal{A}$ for a parameterized minimization problem Π, is defined as a pair of polynomial-time algorithms, called the *reduction algorithm* $\mathcal{R}_\mathcal{A}$ and the *solution lifting algorithm*, that satisfy the following properties:

1. Given an instance (I, k) of Π, $\mathcal{R}_\mathcal{A}$ computes an instance $(I', k') = \mathcal{R}_\mathcal{A}((I, k))$ of Π.

2. Given (I, k), (I', k'), and a solution S' to (I', k'), the solution lifting algorithm produces a solution S to (I, k) such that $\frac{\Pi(I,k,S)}{\mathrm{OPT}(I,k)} \leq \alpha \frac{\Pi(I',k',S')}{\mathrm{OPT}(I',k')}$.

The *size* of a polynomial time pre-processing algorithm $\mathcal{A}$ is defined as, $\mathrm{size}_\mathcal{A}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{R}_\mathcal{A}(I, k), I \in \Sigma^*\}$.

An *α-approximate kernel* for Π is an α-approximate polynomial-time preprocessing algorithm $\mathcal{A}$ such that the size of $\mathcal{A}$ is upper bounded by a function in the input parameter k. In the case of an *α-approximate bikernel*, the reduction algorithm is allowed to output an instance of any problem. However, the size of the instance still needs to be bounded by a function of the input parameter k. A *polynomial-size approximate kernelization scheme (PSAKS)* is a family of α-approximate polynomial kernelization algorithms for each $\alpha > 1$, with size of each bounded by a polynomial in the input parameter. The size of an output instance of a PSAKS, when run on (I, k) with approximation parameter α, must be upper bounded by $f(\alpha)k^{g(\alpha)}$, for some functions f and g independent of $|I|$ and k. We refer the reader to the work of Lokshtanov et al. [LPRS16] for more details and several examples. In this work, we exhibit lossy kernels for Connected Dominating Set (CDS), which is defined as follows, where G denotes the input graph, $k \in \mathbb{N}$ and D is a subset of its vertices.

$$\mathrm{CDS}(G, k, D) = \begin{cases} \infty & \text{if } D \text{ is not a connected dominating set in } G \\ \min\{|D|, k+1\} & \text{otherwise} \end{cases}$$

Notice that in standard kernelization or fixed parameter tractable algorithms, when we parameterize by solution size k, for a decision version of a minimization problem, we actually do not care about solutions of size more than k. However, we always aim for efficient algorithms, where efficiency is measured in terms of k.

Going by the same logic, we set $\mathrm{CDS}(G, k, D) = k + 1$, when $|D| \geq k + 1$, so that all connected dominating sets D of cardinality more than k are "equally bad" or indistinguishable. The symbol ∞ is used to distinguish between actual solutions and other strings which are not solutions.

8.3 Biclique-free graphs

In this section we show that Connected Dominating Set, parameterized by solution size, admits a PSAKS on $K_{d,d}$-free graphs. More precisely, we show that Connected Dominating Set admits a $(1+\epsilon)$-approximate kernel on at most $k^{\mathcal{O}(\frac{d^2}{\epsilon})}$ vertices.

Finding the domination core. We begin by formally introducing the notion of an r-domination core, where $r \in \mathbb{N}$. We restate that all of our results can be easily extended to $K_{i,j}$-free graphs, for constant positive integers i and j.

Definition 8.15 (r-domination core)**.** Let G be a graph and $Z \subseteq V(G)$. We say that Z is an *r-domination core* if every set D of size at most r that dominates Z also dominates $V(G)$.

Note that $V(G)$ is an r-domination core, for every r. However, our objective is to obtain a k-domination core Z whose size is polynomially bounded in k. To that end, we start with $Z = V(G)$ and repeatedly reduce the size of Z, maintaining a k-domination core throughout.

Lemma 8.16. *There exists a polynomial-time algorithm that, given a $K_{d,d}$-free graph G and a k-domination core $Z \subseteq V(G)$ with $|Z| > (2d+1)k^{d+1}$, either correctly concludes that $\mathbf{ds}(G) > k$ (and hence $\mathbf{cds}(G) > k$) or outputs a vertex $z \in Z$ such that $Z \setminus \{z\}$ is a k-domination core.*

Proof. We design such an algorithm as follows. If there is no vertex $v \in V(G)$ such that the cardinality of the neighborhood of v in Z is at least $\lceil \frac{|Z|}{k} \rceil$, then the algorithm terminates and declares that $\mathbf{ds}(G) > k$. Otherwise to find $z \in Z$, the algorithm constructs a sequence of sets $Z = X_0 \supseteq X_1 \supseteq \cdots \supseteq X_\ell$ and a set $S = \{v_1, \ldots, v_\ell\} \subseteq V(G)$ such that $X_i \subseteq N[v_i]$. We construct the sets $Z = X_0 \supseteq X_1 \supseteq \cdots \supseteq X_\ell$ and the set S using an iterative procedure. Initially, we set $S := \emptyset$ and $X_0 := Z$. At step i, if there is a vertex $v_i \in V(G) \setminus S$ whose neighborhood in X_{i-1} has at least $\lceil \frac{|X_{i-1}|}{k} \rceil$ vertices, then add v_i to S and set $X_i := X_{i-1} \cap N[v_i]$. If no such vertex v_i exists, then the algorithm outputs an arbitrary vertex $z \in X_{i-1} \setminus S$ and stops.

The above procedure will construct a sequence $Z = X_0 \supseteq X_1 \supseteq \cdots \supseteq X_\ell$ and a set $S = \{v_1, \ldots, v_\ell\}$. We first claim that $\ell < d$. Suppose $\ell \geq d$. Then there is a complete bipartite subgraph H of G with bipartition $\{v_1, \ldots, v_d\}$ and $X_d \setminus \{v_1, \ldots, v_d\}$. Since $|X_d| \geq \frac{|Z|}{k^d} \geq \frac{(2d+1)k^{d+1}}{k^d} \geq 2d+1$, H contains $K_{d,d}$ as a subgraph, which is a contradiction to the fact that G is $K_{d,d}$-free. Hence $\ell < d$.

Moreover, since $|X_\ell| \geq \frac{|Z|}{(k)^\ell} \geq (2d+1)k^2$, there always exists a vertex $z \in X_\ell \setminus S$ that the algorithm can select to output.

Now we prove the correctness of the algorithm. Clearly, if there is no vertex $v \in V(G)$ such that the cardinality of the neighborhood of v in Z is at least $\lceil \frac{|Z|}{k} \rceil$, then $\mathbf{ds}(G) > k$ and the algorithm declares it correctly. Otherwise let $z \in X_\ell \setminus S$ be the output and let $Z' = Z \setminus \{z\}$. We need to prove that Z' is still a k-domination core. Let D be a set of size at most k that dominates Z'. To prove that D is a dominating set in G, it is enough to show that D dominates Z (because Z is a k-domination core). Notice that every vertex in S is adjacent to z. Hence, to show that D also dominates Z, it is enough to show that $D \cap S \neq \emptyset$. We will show that if $D \cap S = \emptyset$, then D cannot dominate all of $X_\ell \setminus \{z\} \subseteq Z'$. Since our algorithm stops at step $\ell + 1$, we know that every vertex outside S can dominate strictly less than $(2d+1)k^{d-\ell}$ vertices from X_ℓ. As $|X_\ell \setminus \{z\}| \geq (2d+1)k^{d+1-\ell}$, $|D| \leq k$ and every vertex outside S can dominate at most $(2d+1)k^{d-\ell} - 1$ vertices of X_ℓ, we have that $D \cap S \neq \emptyset$. This completes the proof of the lemma. □

Reducing connectors and dominators. Armed with a k-domination core Z of size at most $(2d+1)k^{d+1}$, we partition the graph into two sets Z and $R = V(G) \setminus Z$. Next, we define the equivalence relation $\simeq$ on R as follows: $\forall u, v \in R, u \simeq v$ if and only if $N_Z(u) = N_Z(v)$. That is, we partition vertices in R according to their neighborhoods in Z. Let $\mathcal{R}$ be the set of equivalence classes defined by $\simeq$. Our goal is to bound the size of $\mathcal{R}$. To that end, we construct a bipartite graph H with bipartition (A, B) from G as follows. We add a vertex $a_z \in A$ for each vertex $z \in Z$ and we add a vertex $b_\kappa \in B$ for each equivalence class κ of relation $\simeq$. We add an edge $a_z b_\kappa \in E(H)$ whenever z is in $N_Z(w)$, for some $w \in \kappa$. Finally, delete any isolated vertices in H. It is not hard to see that H is a subgraph of G and hence is $K_{d,d}$-free. As a direct implication of Lemma 8.12, we can bound the size of B by $\mathcal{O}(k^{\mathcal{O}(d^2)})$.

Lemma 8.17. *The equivalence relation $\simeq$ has at most $k^{\mathcal{O}(d^2)}$ classes, i.e. $|\mathcal{R}| \in k^{\mathcal{O}(d^2)}$.*

As Z is a k-domination core, to find a dominating set of size at most k it is enough to find a set which dominates Z. Hence for the purpose of domination, it is redundant to pick more than one vertex from an equivalence class in $\mathcal{R}$. However, to get a connected dominating set, we may need to choose more than one vertex from an equivalence class. The following lemma finds a small set of relevant vertices which "approximately" preserves the connectivity requirements.

Lemma 8.18. *Let (G, k) be an instance of* Connected Dominating Set*, where G is a connected and $K_{d,d}$-free graph. For any fixed $\epsilon > 0$, there is a polynomial-time algorithm that either correctly concludes that $\mathbf{cds}(G) > k$ or outputs a set $Y \subseteq V(G)$ of cardinality $k^{\mathcal{O}(\frac{d^2}{\epsilon})}$ such that (i) Y contains a k-domination core of G and (ii) $OPT(G[Y], k) \leq (1+\epsilon)OPT(G, k)$.*

Proof. Let $t \geq 1$ be a constant, which we fix later. We first define the Group Steiner Tree problem. The input consists of an n-vertex graph H and sets

$Q_1, Q_2, \ldots, Q_t \subseteq V(H)$ called *groups*. The task is to find a tree T of minimum size which contains at least one vertex from every group Q_i. The Group Steiner Tree problem can be solved in $\mathcal{O}(2^t \cdot n^{\mathcal{O}(1)})$-time using polynomial space [MPR+10].

Now we design an algorithm $\mathcal{A}$ with the properties claimed in the lemma. Algorithm $\mathcal{A}$ will apply Lemma 8.16 repeatedly starting from a k-domination core $V(G)$ and either conclude that $\mathbf{cds}(G) > k$ or find a k-domination core Z of size at most $(2d+1)k^{d+1}$ (in polynomial time). Now, let $\mathcal{Z}$ be a family of groups $\{\{z\} \mid z \in Z\}$ and let $\mathcal{R}$ be the set of equivalence classes defined by $\simeq$. The set $\mathcal{R} \cup \mathcal{Z}$ forms a family of groups of vertices in $V(G)$. For every subset $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\} \subseteq \mathcal{R} \cup \mathcal{Z}$ of size at most $2t$ of groups in $\mathcal{R} \cup \mathcal{Z}$, construct a Group Steiner Tree instance on the graph G with groups $Q_1, \ldots, Q_\ell$. Let $T_{\mathcal{Q}}$ be the corresponding solution. Note that since t is a constant each instance can be solved in polynomial time. For every instance that we solve, if the size of $T_{\mathcal{Q}}$ is at most $2t$ then we mark the vertices of $T_{\mathcal{Q}}$ in G. We denote the set of all marked vertices by $\bigcup T_{\mathcal{Q}}$. If $\bigcup T_{\mathcal{Q}}$ is not a dominating set in G, then algorithm $\mathcal{A}$ declares that $\mathbf{cds}(G) > k$. Otherwise, since G is assumed to be connected, algorithm $\mathcal{A}$ runs the polynomial-time algorithm mentioned in Proposition 8.1 to obtain a set $W \subseteq V(G)$ such that $\bigcup T_{\mathcal{Q}} \cup W$ is a connected dominating set in G and $|\bigcup T_{\mathcal{Q}} \cup W| \leq 3|\bigcup T_{\mathcal{Q}}|$. Algorithm $\mathcal{A}$ outputs $Y = \bigcup T_{\mathcal{Q}} \cup W$ as the required set, where $G[Y]$ is a connected graph. Note that we also marked a solution of Group Steiner Tree for $\mathcal{Q} = \{Q\}$ for every $Q \in \mathcal{R} \cup \mathcal{Z}$. Hence $\bigcup T_{\mathcal{Q}}$ contains all of Z and a vertex from each group in $\mathcal{R}$.

We now prove the correctness of the algorithm. Suppose $\mathbf{cds}(G) \leq k$, then we claim that algorithm $\mathcal{A}$ outputs a set Y. If $\mathbf{cds}(G) \leq k$, then by Lemma 8.16, algorithm $\mathcal{A}$ will, as the first step, correctly find a k-domination core Z of size at most $(2d+1)k^{d+1}$ and the set of groups $\mathcal{R} \cup \mathcal{Z}$. Let D be the graph induced by a connected dominating set of size at most k in G (i.e. D is a tree). Let D' be an arbitrary set of cardinality at most $|V(D)|$ such that for any $w \in V(D)$, there is a vertex $w' \in D'$ with the property that $\{w, w'\} \subseteq Q \in \mathcal{R} \cup \mathcal{Z}$ and $w' \in \bigcup T_{\mathcal{Q}}$. That is, if $V(D)$ has a vertex from a group Q in $\mathcal{R} \cup \mathcal{Z}$, then D' also has a vertex from group Q which is, in addition, marked by algorithm $\mathcal{A}$. Note that we can construct the set D' since $\bigcup T_{\mathcal{Q}}$ contains at least one vertex from each group in $\mathcal{R} \cup \mathcal{Z}$.

Claim 8.19. *D' is a dominating set in G.*

Proof of the Claim. Notice that $Z \cap V(D) = Z \cap D'$ and if any vertex in Z is adjacent to a vertex in a group Q, then it is adjacent to all vertices in group Q. This implies that D' also dominates Z and since $|D'| \leq |V(D)| \leq k$, by the definition of a k-domination core, D' is a dominating set in G. □

Hence we can conclude that if $\mathbf{cds}(G) \leq k$, then $\bigcup T_{\mathcal{Q}}$ always dominates G and algorithm $\mathcal{A}$ outputs a set Y. Moreover, since $Z \subseteq \bigcup T_{\mathcal{Q}} \subseteq Y$, we have that Y contains a k-domination core. This concludes the proof of property (i) of the lemma.

Now we prove that $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$. Let D^* be the graph induced by a connected dominating set for G of minimum cardinality (i.e. D^* is a tree). If $|V(D^*)| > k$, then $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$ holds trivially. So we assume that $|V(D^*)| \leq k$. We let $\mathcal{F}(D^*,t) = \{T_1, T_2, \cdots\}$ denote a (D^*,t)-covering family. Proposition 8.3 implies that there exists such a family for which $|\mathcal{F}(D^*,t)| \leq \frac{|V(D^*)|}{t} + 1$ and $\sum_{T\in\mathcal{F}(D^*,t)} |V(T)| \leq (1+\frac{1}{t})|V(D^*)| + 1$. Moreover, the size of each connected subgraph T (in this case also subtree) is at most $2t$. We construct a new family $\mathcal{F}'$ from $\mathcal{F}(D^*,t)$ as follows. For each $T \in \mathcal{F}(D^*,t)$, we replace T by $T_{\mathcal{Q}}$, where $\mathcal{Q}$ is the set of groups from $\mathcal{R}\cup\mathcal{Z}$ such that $Q \in \mathcal{Q}$ if and only if $V(T)\cap Q \neq \emptyset$ and $T_{\mathcal{Q}}$ is the set of marked vertices in an optimal Steiner tree connecting vertices from the groups in $\mathcal{Q}$. Note that the fact that T is of size at most $2t$ guarantees the existence of $T_{\mathcal{Q}}$ (by construction). Moreover, the size of $T_{\mathcal{Q}}$ is at most the size of T, since T is also a solution for Group Steiner Tree for $\mathcal{Q}$. Let $D_{\mathcal{F}'}$ denote the union of all vertices in $\mathcal{F}'$. Let D' be a subset of $D_{\mathcal{F}'}$, of cardinality at most $|V(D^*)|$, such that for any $w \in V(D^*)$, there is a vertex $w' \in D'$ with the property that $\{w,w'\} \subseteq Q \in \mathcal{R}\cup\mathcal{Z}$ and $w' \in D_{\mathcal{F}'}$. That is, if $V(D^*)$ has a vertex from a group Q in $\mathcal{R}\cup\mathcal{Z}$, then D' also has a vertex from group Q. Using the same arguments as in the proof of Claim 8.19, we know that D' is a dominating set in G. This implies that $D_{\mathcal{F}'} \supseteq D'$ is also a dominating set in G. Applying Proposition 8.1 in $G[Y]$ (with $D_{\mathcal{F}'}$ as dominator and since $G[Y]$ is connected), we obtain a connected dominating set of size at most $2|\mathcal{F}(D^*,t)| + |D_{\mathcal{F}'}| \leq \frac{2|V(D^*)|}{t} + 2 + (1+\frac{1}{t})|V(D^*)| + 1 = (1+\frac{3}{t})|V(D^*)| + 3$. Now we can fix the constant t appropriately (i.e. roughly $\frac{3}{\epsilon}$) and we get that $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$.

Now we show that $|Y| = k^{\mathcal{O}(d^2)}$. By Lemmata 8.16 and 8.17, we have that $|\mathcal{R}\cup\mathcal{Z}| = \mathcal{O}(k^{\mathcal{O}(d^2)})$. From the construction, it follows that $|\bigcup T_{\mathcal{Q}}| = \mathcal{O}(2t|\mathcal{R}\cup\mathcal{Z}|^{\mathcal{O}(t)}) = \mathcal{O}(2t\cdot k^{\mathcal{O}(td^2)}) = k^{\mathcal{O}(\frac{d^2}{\epsilon})}$. Notice that $Y = \bigcup T_{\mathcal{Q}} \cup W$, where W is obtained by applying Proposition 8.1 and hence we have that $Y = |\bigcup T_{\mathcal{Q}} \cup W| \leq 3|\bigcup T_{\mathcal{Q}}| = k^{\mathcal{O}(\frac{d^2}{\epsilon})}$. □

Theorem 28. *For every $\epsilon > 0$,* Connected Dominating Set*, parameterized by solution size, admits a $(1+\epsilon)$-approximate kernel with $k^{\mathcal{O}(\frac{d^2}{\epsilon})}$ vertices on $K_{d,d}$-free graphs.*

Proof. Let (G,k) be the input instance, where G is a connected $K_{d,d}$-free graph and k is a positive integer. We first describe the reduction algorithm $\mathcal{R}_{\mathcal{A}}$. Using the algorithm of Lemma 8.18, with inputs (G,k) and ϵ, in polynomial time, algorithm $\mathcal{R}_{\mathcal{A}}$ either concludes that $\mathbf{cds}(G) > k$ and outputs $((\{v\},\emptyset),0)$; or obtains a set $Y \subseteq V(G)$ and outputs $(G[Y],k)$ as the reduced instance. Let (G',k') be the reduced instance. When $G' = (\{v\},\emptyset)$, the size of the reduced instance is a constant. Otherwise, by Lemma 8.18, we know that $|Y| = k^{\mathcal{O}(\frac{d^2}{\epsilon})}$, which bounds the kernel size.

The solution lifting algorithm works as follows. Given a solution D' to the instance (G',k'), if D' is a connected dominating set of size at most k, then the algorithm returns the set D'. If D' is a connected dominating set of size greater

than k, then the solution lifting algorithm returns $V(G)$ as a solution to (G, k). If D' is not a connected dominating set of G', then the solution lifting algorithm will output $\emptyset$. Let D be the output of the solution lifting algorithm.

We prove that the above reduction algorithm together with the solution lifting algorithm constitute a $(1+\epsilon)$-approximate kernel. Note that if D' is not a valid solution of G', then $\emptyset$ is not a valid solution for G and $CDS(G', k', D') = CDS(G, k, D) = \infty$. Hence we can restrict ourselves to the case when D' is a connected dominating set of G'. First, consider the case where the algorithm of Lemma 8.18 outputs $Y \subseteq V(G)$ and the reduced instance is hence $(G', k') = (G[Y], k)$. From Lemma 8.18, we have that $OPT(G[Y], k) \leq (1+\epsilon)OPT(G, k)$. We show that in this case $CDS(G, k, D) = CDS(G', k', D')$. If $|D'| > k$, then $CDS(G, k, D) = CDS(G, k, V(G)) = k + 1 = CDS(G', k', D')$. So assume that $|D'| \leq k$, which implies $D = D'$. Since D' is a connected dominating set of $G[Y]$ and, by Lemma 8.18, Y contains a k-domination core of G, it follows that D' dominates G and $CDS(G, k, D) = CDS(G', k', D')$. Combining $CDS(G, k, D) = CDS(G', k', D')$ and $OPT(G[Y], k) \leq (1+\epsilon)OPT(G, k)$ we get $\frac{CDS(G,k,D)}{OPT(G,k)} \leq (1+\epsilon)\frac{CDS(G',k',D')}{OPT(G',k')}$. When $(G', k') = ((\{v\}, \emptyset), 0)$, we can easily verify that the above mentioned approximation guarantee holds. □

8.4 Graphs of bounded expansion

In this section we show that Connected Dominating Set, parameterized by solution size, admits a $(1+\epsilon)$-approximate bikernel on at most $\mathcal{O}(f(\epsilon) \cdot k)$ vertices. In fact, the reduced instance will be an instance of Subset Connected Dominating Set (SCDS), which is defined as follows:

$$\text{SCDS}((G, S), k, D) = \begin{cases} \infty & \text{if } D \text{ is not a connected } S\text{-dominator in } G \\ \min\{|D|, k+1\} & \text{otherwise} \end{cases}$$

The first phase of our algorithm, i.e. finding a domination core, closely follows the work of Drange et al. [DDF+16] but requires subtle changes. We fix a graph class $\mathcal{G}$ that has bounded expansion and let (G, k) be the input instance of CDS, where $G \in \mathcal{G}$ and G is connected. We assume that $\nabla_0(\mathcal{G}) \geq 1$, otherwise G is a forest and the problem can be solved in linear time.

8.4.1 Finding the domination core

We begin by formally introducing the notion of a c-exchange domination core, which is different from the definition used in the previous section and from the one considered in [DDF+16]. Here, c is a fixed constant which we set later.

Definition 8.20 (c-exchange domination core)**.** Let G be a graph and Z be a subset of vertices of G. We say that Z is a *c-exchange domination core* if for every set X that dominates Z one of the following conditions holds:

1. X dominates G, or

2. there exist $A \subseteq X$ and $B \subseteq V(G)$ such that $|B| < |A| \leq c$ and $(X \setminus A) \cup B$ is a set that dominates Z. Moreover the number of connected components of $(X \setminus A) \cup B$ is at most the number of connected components of X. In particular, if X is a connected set then $(X \setminus A) \cup B$ is also a connected set.

Proposition 8.21. *Let G be a graph, c be a constant, and Z be a c-exchange domination core of G, and $X \subseteq V(G)$ be a Z-dominator. Then, there is a set Y such that (i) $|Y| \leq |X|$, (ii) Y dominates G, (iii) Y can be computed from X in polynomial time, and (iv) the number of connected components of Y is at most the number of connected components of X.*

Proof. By Definition 8.20, if X dominates G then we are done. Otherwise, there exist $A \subseteq X$ and $B \subseteq V(G)$ such that $|B| < |A| \leq c$, $X' = (X \setminus A) \cup B$ is a set that dominates Z, and the number of connected components of X' is at most the number of connected components of X. Moreover, we can find such sets A and B by going through all possibilities in time $\mathcal{O}(|V(G)|^{2c-1})$ (i.e., in polynomial time for fixed c). By applying this argument iteratively on X', we will eventually find a set Y which dominates G. Furthermore, by Definition 8.20, $|B| < |A|$, and hence the size of the Z-dominator drops by one in each step. Therefore, the required set Y can be found in time $\mathcal{O}(|V(G)|^{2c})$. □

Clearly, $V(G)$ is a c-exchange domination core, for any c, but we look for a c-exchange domination core that is linear in k. Hence, we start with $Z = V(G)$ and gradually reduce $|Z|$ by removing one vertex at a time, while maintaining the invariant that Z is a c-exchange domination core. To this end, we need to prove Lemma 8.22. Note that we only remove vertices from Z at this stage (no vertex deletions), and hence the graph remains intact.

Lemma 8.22. *There exists a constant $C_{\mathbf{core}} > 0$ depending only on a fixed (finite) number of grads of $\mathcal{G}$ and a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$ and a c-exchange domination core $Z \subseteq V(G)$ with $|Z| > C_{\mathbf{core}} \cdot k$, either correctly concludes that $\mathbf{cds}(G) > k$ or finds a vertex $z \in Z$ such that $Z \setminus \{z\}$ is still a c-exchange domination core.*

The rest of the subsection is dedicated to proving Lemma 8.22. The algorithm of Lemma 8.22 consists of building a structural decomposition of the graph G. More precisely, we identify a small set X that dominates G, so that if X was deleted from the graph, Z would contain a large subset S, which is 2-scattered in the remaining graph. Given such a structure, we can argue that in any optimal Z-dominator, vertices of X serve as dominators for almost all the vertices of S. This is because any vertex of $V(G) \setminus X$ can dominate at most one vertex from S. Since S will be large compared to X, some vertices of S will be indistinguishable from the point of view of domination via X, and these will be precisely the vertices that can be removed from the domination core. The following lemma, which was proved by Drange et al. [DDF+16], gives us such a decomposition.

Lemma 8.23. *There exists a constant $C'_{\boldsymbol{Z}} > 0$ depending only on a fixed (finite) number of grads of $\mathcal{G}$ and a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, an integer k, a constant $C'_{\boldsymbol{S}} > 0$, and a set $Z' \subseteq V(G)$ with $|Z'| > C'_{\boldsymbol{Z}} \cdot k$, either correctly concludes that $\mathbf{ds}(G) > k$ or finds a pair (X', S') with the following properties: (1) $|X'| \leq C'_{\boldsymbol{X}} \cdot k$, (2) X' is a Z'-dominator in G, (3) for each $u \in V(G) \setminus X'$ we have $|M_3^G(u, X')| \leq C'_{\boldsymbol{M}}$, and (4) $S' \subseteq Z' \setminus X'$ is 2-scattered in $G - X'$, and $|S'| \geq C'_{\boldsymbol{S}} \cdot |X|$, where $C'_{\boldsymbol{X}}$ and $C'_{\boldsymbol{M}}$ are constants depending only on a fixed number of grads of $\mathcal{G}$.*

Corollary 8.24. *There exists a constant $C_{\boldsymbol{Z}} > 0$ depending only on a fixed (finite) number of grads of $\mathcal{G}$ and a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, an integer k, a constant $C_{\boldsymbol{S}} > 0$, and a c-exchange domination core $Z \subseteq V(G)$ with $|Z| > C_{\boldsymbol{Z}} \cdot k$, either correctly concludes that $\mathbf{ds}(G) > k$ or finds a pair (X, S) with the following properties: (1) $|X| \leq C_{\boldsymbol{X}} \cdot k$, (2) X dominates G, (3) for each $u \in V(G) \setminus X$ we have $|M_3^G(u, X)| \leq C_{\boldsymbol{M}}$, and (4) $S \subseteq Z \setminus X$ is 2-scattered in $G - X$, and $|S| \geq C_{\boldsymbol{S}} \cdot |X|$, where $C_{\boldsymbol{X}}$ and $C_{\boldsymbol{M}}$ are constants depending only on a fixed number of grads of $\mathcal{G}$.*

Proof. First we run the algorithm of Lemma 8.23 for G, k, $Z' = Z$, $C'_{\mathbf{Z}} = C_{\mathbf{Z}}$, and constant $C'_{\mathbf{S}} \geq C_{\mathbf{S}}$, which we fix later in the proof. If the algorithm of Lemma 8.23 concludes that $\mathbf{ds}(G) > k$, we correctly conclude that $\mathbf{ds}(G) > k$. Otherwise, let (X', S') be the output of the algorithm of Lemma 8.23. Since Z is a c-exchange domination core, we find using Proposition 8.21, a set Y, such that $|Y| \leq |X'|$ and Y dominates G. We let $X = \mathrm{cl}_3(Y \cup X')$ and $S = S' \setminus X$. Since X is a superset of Y, X satisfies property (2). Note that by Lemma 8.9, X can be computed in polynomial time and it satisfies property (3). Moreover, $|X| = \mathrm{cl}_3(Y \cup X') \leq C_{\mathbf{cl1}} \cdot |Y \cup X'| \leq 2C_{\mathbf{cl1}} \cdot |X'| \leq 2C_{\mathbf{cl1}} \cdot C'_{\mathbf{X}} \cdot k$ (Lemma 8.9) and property (1) follows with $C_{\mathbf{X}} = 2C_{\mathbf{cl1}} \cdot C'_{\mathbf{X}}$. Hence $|S| \geq |S'| - |X| \geq C'_{\mathbf{S}} \cdot |X'| - 2C_{\mathbf{cl1}} \cdot |X'| \geq (C'_{\mathbf{S}} - 2C_{\mathbf{cl1}}) \cdot |X'| \geq \frac{(C'_{\mathbf{S}} - 2C_{\mathbf{cl1}})}{2C_{\mathbf{cl1}}} |X|$. Therefore, if we set the constant $C'_{\mathbf{S}} = 2C_{\mathbf{cl1}}(C_{\mathbf{S}} + 2C_{\mathbf{cl1}})$, we satisfy all properties. □

Note that when $\mathbf{ds}(G) > k$ we can also conclude that $\mathbf{cds}(G) > k$. Hence, in the rest of the section we assume that we are given G, Z, and the constructed sets X and S. We let $R = V(G) \setminus X$. Using this notation, S is 2-scattered in the graph $G[R]$. Recall that for any vertex $u \in R$, we have $|M_3(u, X)| \leq C_{\mathbf{M}}$. Define the following equivalence relation $\simeq$ on S: for $u, v \in S$, $u \simeq v \quad \Leftrightarrow \quad M_i(u, X) = M_i(v, X)$ for each $1 \leq i \leq 3$.

Lemma 8.25. *There exists a constant $C_{\boldsymbol{eq}} > 0$ depending only on a fixed (finite) number of grads of $\mathcal{G}$ such that equivalence relation $\simeq$ has at most $C_{\boldsymbol{eq}} \cdot 3^{C_{\boldsymbol{eq}}} \cdot |X|$ classes.*

Proof. From Lemma 8.10, we know that the number of different 3-projections in X of vertices of R is bounded by $C_{\mathbf{ex}} \cdot |X|$. Observe that for each $u \in S$, we have $M_1(u, X) \subseteq M_2(u, X) \subseteq M_3(u, X)$ and the number of choices for $M_3(u, X)$ is again at most $C_{\mathbf{ex}} \cdot |X|$ (since $S \subseteq R$). Moreover, since $u \in R$, we have that $|M_3(u, X)| \leq C_{\mathbf{M}}$ (property (3) of Corollary 8.24). Hence, to define sets $M_i(u, X)$

for $1 \leq i < 3$ it suffices, for every $w \in M_3(u, X)$, to choose the smallest index j, $1 \leq j \leq 3$, such that $w \in M_j(u, X)$. The number of such choices is at most $3^{C_{\mathbf{M}}}$, and hence the claim follows by setting $C_{\mathbf{eq}} = C_{\mathbf{ex}} + C_{\mathbf{M}}$. □

We can now set the constant $C_{\mathbf{S}}$ that is required in Corollary 8.24 and the constant c of the exchange domination core. We let $C_{\mathbf{S}} = (4C_{\mathbf{eq}} + 1) \cdot C_{\mathbf{eq}} \cdot 3^{C_{\mathbf{eq}}}$ and $c = (4C_{\mathbf{eq}} + 1)$. Since we have that $|S| > C_{\mathbf{S}} \cdot |X|$, from Lemma 8.25 and the pigeonhole principle we infer that there is a class κ of relation $\simeq$ with $|\kappa| > 4C_{\mathbf{eq}}+1$. Note that we can find such a class κ in polynomial time, by computing the classes of $\simeq$ directly from the definition and examining their sizes. We are ready to prove the final lemma of this section: any vertex of κ can be removed from the c-exchange domination core Z (recall that $S \subseteq Z$), which concludes the proof of Lemma 8.22.

Lemma 8.26. *Let $z \in \kappa$. Then $Z \setminus \{z\}$ is a c-exchange domination core.*

Proof. Let $D \subseteq V(G)$ be a set that dominates $Z' = Z \setminus \{z\}$ such that D does not satisfy condition (2) of Definition 8.20 with respect to Z' (as otherwise we are done). In particular, this implies that D cannot satisfy condition (2) with respect to Z either. In other words, there exist no sets $A \subseteq D$ and $B \subseteq V(G)$ such that $|B| < |A| \leq c$, $(D \setminus A) \cup B$ has at most as many connected components as D, and $(D \setminus A) \cup B$ dominates Z. Therefore, if D dominates all of Z, then D has to satisfy condition (1) and dominate G. Hence, z is not dominated by D. We prove that this leads to a contradiction.

Every vertex $s \in \kappa \setminus \{z\}$ is dominated by D (since $\kappa \setminus \{z\} \subseteq Z'$). For each such s, let $v(s)$ be an arbitrarily chosen vertex of D that dominates s. If $v(s) \in X$, then $v(s)$ also dominates z, since $M_1(s, X) = M_1(z, X)$. Consequently, $v(s) \notin X$ (because D does not dominate Z) and the vertices $v(s)$ are pairwise different for all $s \in \kappa \setminus \{z\}$, as S (and κ) is a 2-scattered set. Let $W' = \{v(s) : s \in \kappa \setminus \{z\}\}$. Since $|\kappa| > 4C_{\mathbf{eq}} + 1$, we have $|W'| \geq c = 4C_{\mathbf{eq}} + 1$. Let W be a set of arbitrarly chosen c vertices of W'. Define $D' = (D \setminus W) \cup M_3(z, X)$.

We first show that D' dominates Z'. Towards that, it is enough to show that every vertex in $N[W]$ is dominated by $M_3(z, X) \subseteq D'$. By property (2) of Corollary 8.24, X is a dominating set in G. Hence, every vertex in $N[W] = N(W) \cup W$ either belongs to X or has a neighbor in X. Let $Y \subseteq X$ be the set of vertices in X dominating $N[W]$. Since every vertex in $N[W]$ is at distance at most two from some vertex in κ (because each vertex in W is adjacent to a vertex in κ), all vertices in Y are at distance at most three from some vertex in κ. This implies that $Y \subseteq M_3(z, X)$ and therefore D' dominates Z'.

Note that $|M_3(z, X)| < |W| \leq c$ and therefore D', which still dominates Z', violates condition (2) of Definition 8.20. However, the set D' can have more connected components than D has. Therefore, we add additional vertices to D' to ensure connectivity and still violate condition (2). For every vertex $u \in M_3(z, X)$ there is a path P_u of length at most three between u and z. Let $P = \bigcup_{u \in M_3(z,X)} P_u$ and consider the set $D'' = D' \cup P$. Note that $|P| \leq 3|M_3(z, X)| \leq 3C_{\mathbf{eq}}$.

It remains to show that D'' has at most as many connected components as D. If a component C in D contains a vertex from W, then each connected component C' of $C \setminus W$ contains a vertex $v_{C'} \in N(W)$ and as mentioned before, $v_{C'}$ is adjacent to a vertex in $M_3(z, X)$. But in D'' all vertices of $M_3(z, X)$ are in one component due to the vertices of P. Hence all such components of D contribute to only one component of D''. On the other hand, if C does not contain a vertex in W, then clearly all vertices of C are also in D'' and hence C is connected in D'' as well. It follows that D'' has at most as many connected components as D, D'' dominates Z', and $|M_3(z, X) \cup P| \leq C_{\mathbf{eq}} + 3C_{\mathbf{eq}} < |W| = c = 4C_{\mathbf{eq}} + 1$, contradicting the fact that D did not satisfy condition (2) of Definition 8.20. □

8.4.2 Reducing connectors and dominators

Armed with a c-exchange domination core Z whose size is linear in k, our next goal is to reduce the number of connectors and dominators (the number of vertices in $V(G) \setminus Z$). To that end, we need the following lemma which is a generalized version of Lemma 2.11 in [DDF+16].

Lemma 8.27 (Trees closure lemma)**.** *Let $\mathcal{G}$ be a class of bounded expansion and let q and r be positive integers. Let $G \in \mathcal{G}$ be a graph and $X \subseteq V(G)$. Then a superset of vertices $X' \supseteq X$ can be computed in polynomial time, with the following properties: (1) For every $Y \subseteq X$ of size at most q, if $\mathbf{st}_G(Y) \leq rq$ then $\mathbf{st}_{G[X']}(Y) = \boldsymbol{st}_G(Y)$. (2) $|X'| \leq C_{\boldsymbol{tc}} \cdot |X|$, where $C_{\boldsymbol{tc}}$ is a constant depending only on r, q, and a finite number of grads of $\mathcal{G}$.*

Proof. First, using Lemma 8.9 we compute $X_0 = \mathrm{cl}_{rq}(X)$. Then, $|X_0| \leq C_{\mathbf{cl1}} \cdot |X|$ and for each vertex $u \notin X_0$ we have $|M^G_{rq}(u, X_0)| \leq C_{\mathbf{cl2}}$. Now, for each set $Y \subseteq X_0$ of at most q vertices, compute an optimal Steiner tree T_Y whose edges do not belong to $G[X_0]$; in case there is no such tree, set $T_Y = \emptyset$. Note that T_Y can be computed in polynomial time for any fixed q [BHKK07]. Define X' to be X_0 plus the vertex sets of all trees T_Y that have size at most rq.

Claim 8.28. *$|X'| \leq C_{\boldsymbol{tc}} \cdot |X_0|$, where $C_{\boldsymbol{tc}}$ is a constant depending only on r, q, and a finite number of grads of $\mathcal{G}$.*

Proof of the Claim. Let H be a graph on vertex set X_0, where $uv \in E(H)$ if and only if there exists Y such that $\{u, v\} \subseteq Y$, $T_Y \neq \emptyset$ and has size at most rq, and hence its vertex set was added to X. Note that we do not add multiedges. For every such set Y, $H[Y]$ induces a clique in H. Let $\omega(H)$ denote the number of cliques in H. Clearly $|X'| \leq |X_0| + rq \cdot \omega(H)$, so it suffices to prove an upper bound on $\omega(H)$.

Consider an edge $uv \in E(H)$. The existence of this edge implies that u and v appear together in some tree T_Y of size at most rq. Since T_Y does not contain any edges from $G[X_0]$ (by construction), there must exist a path $P_{u,v}$ of length at most rq connecting u and v. The internal vertices of $P_{u,v}$ do not belong to X_0. Take any $w \in X' \setminus X_0$, and consider for how many pairs $\{u, v\} \subseteq X_0$ it can hold that $w \in P_{u,v}$. If $\{u, v\}$ is such a pair, then in particular $u, v \in M^G_{rq}(w, X_0)$.

But we know that $|M^G_{rq}(w, X_0)| \leq C_{\mathbf{cl2}}$, so the number of such pairs is at most $\tau \leq (C_{\mathbf{cl2}})^2$. Consequently, we observe that graph H is an $(rq-1)$-shallow minor of $G \odot K_\tau$: when each vertex $w \in X' \setminus X_0$ is replaced with τ copies, then we can realize all the paths between u and v, in $G \odot K_\tau$, so that they are internally vertex-disjoint. From Lemma 8.8, we know that $\nabla_{rq-1}(G \odot K_\tau)$ is bounded polynomially in $\nabla_{rq-1}(G)$ and τ, which in turn is also bounded polynomially in $\nabla_{rq-1}(\mathrm{mc}\, G)$. Hence $\nabla_{rq-1}(G \odot K_\tau)$ is bounded polynomially in $\nabla_{rq-1}(\mathrm{mc}\, G)$. The number of cliques in graph of bounded expansion is linear in the number of vertices [BLS99]. Combining the fact that H has bounded expansion with $|X'| \leq |X_0| + rq \cdot \omega(H)$, the claim follows. □

Claim 8.29. *If $Y \subseteq X_0$ has size at most q and $\mathbf{st}_G(Y) \leq rq$ then $\mathbf{st}_{G[X']}(Y) = \boldsymbol{st}_G(Y)$.*

Proof of the Claim. Let T_Y be an optimal Steiner tree for Y in G, and let $T_1, T_2, \ldots, T_p$ be the subtrees of size greater than one obtained after deleting all edges of T_Y for which both endpoints are in X_0. Note that deleting such edges can only create either singleton vertices or subtrees of size greater than one. Moreover, let Y_i, $1 \leq i \leq p$, denote the set $Y \cap V(T_i)$. The existence of T_i certifies that some tree of size at most $|T_i|$ was added when constructing X' from X_0, and hence $\mathbf{st}_{G[X']}(Y_i) \leq |T_i|$. Consequently, we infer that

$$\mathbf{st}_{G[X']}(Y) \leq \sum_{i=1}^{p} \mathbf{st}_{G[X']}(Y_i) + |Y \setminus \bigcup_{i=1}^{p} Y_i| \leq \sum_{i=1}^{p} |T_i| + |Y \setminus \bigcup_{i=1}^{p} Y_i| \leq |T_Y| = \mathbf{st}_G(Y).$$

The opposite inequality $\mathbf{st}_{G[X']}(Y) \geq \mathbf{st}_G(Y)$ follows directly from the fact that $G[X']$ is an induced subgraph of G. □

Claim 8.28 and the fact that $|X_0| \leq C_{\mathbf{cl1}}|X|$ prove property (2). Claim 8.29 and the fact that $X \subseteq X_0$ prove property (1). □

Now let $\dot{Z}$ be a superset of Z, which we will fix later in Lemma 8.32, such that $|\dot{Z}| = \mathcal{O}(k)$. We compute $Z' = \mathrm{cl}_1(\dot{Z})$ using Lemma 8.9; then we have that $|Z'| = \mathcal{O}(|\dot{Z}|) = \mathcal{O}(k)$. Partition $V(G) \setminus Z'$ into equivalence classes with respect to the following relation $\simeq$: For $u, v \in V(G) \setminus Z'$, set: $u \simeq v \quad \Leftrightarrow \quad M_1(u, Z') = M_1(v, Z')$. From Lemma 8.9 we know that for each $u \in V(G) \setminus Z'$, it holds that $|M_1(u, Z')| = \mathcal{O}(1)$. Moreover, Lemma 8.10 implies that the number of possible different projections $M_1(u, Z')$ for $u \in V(G) \setminus Z'$ is at most $\mathcal{O}(|Z'|)$. Hence, using the same reasoning as in the proof of Lemma 8.25 we obtain the following.

Lemma 8.30. *The equivalence relation $\simeq$ has at most $\mathcal{O}(|Z'|)$ classes.*

Construct a graph $\ddot{G}$ as follows. Start with G and, for each equivalence class κ of relation $\simeq$, add a new vertex u_κ which is connected to all vertices in κ. Let $U = \bigcup_{\kappa \in \simeq} u_\kappa$. Our next step is to apply Lemma 8.27 to graph $\ddot{G}$ with $X = Z' \cup U$, $r = 2$, and $q = 2t$ (we fix the value of t later). Before we do so, we need to show that $\ddot{G}$ still has bounded expansion.

Lemma 8.31. *The graph $\ddot{G}$ has bounded expansion.*

Proof. We first construct a bipartite graph H with bipartition (A, B) as follows. We add a vertex $x_z \in A$ for each vertex $z \in Z'$ and we add one vertex $y_\kappa \in B$ for each equivalence class κ of relation $\simeq$. We add an edge $x_z y_\kappa \in E(H)$ whenever z is in $M_1(w, Z')$, for some $w \in \kappa$. Finally, delete any isolated vertices in H (applies if some equivalence class has no neighbors in Z'). It is not hard to see that H is a subgraph of G and, consequently, has bounded expansion. We can therefore apply Lemma 8.11 to obtain a mapping $\varphi : B \to A$ with $y\varphi(y) \in E(H)$, for each $y \in B$, and $\varphi^{-1}(x_z) \leq C_{\mathbf{ch}}$, for each $x_z \in A$.

We now consider the graph $\dddot{G}$ which is obtained by adding a universal vertex to $G \odot K_{C_{\mathbf{ch}}}$. From Lemma 8.8, we know that $G \odot K_{C_{\mathbf{ch}}}$ has bounded expansion. Applying Lemma 8.7 to $\ddot{G}$, we know that $\dddot{G}$ also has bounded expansion. Hence, it remains to show that $\ddot{G}$ is a subgraph of $\dddot{G}$. From the mapping φ, we can associate each (except at most one) equivalence class with some vertex in Z'. In addition, every vertex in Z' is associated with at most $C_{\mathbf{ch}}$ classes. In other words, when each vertex in Z' is replaced by $C_{\mathbf{ch}}$ copies, each equivalence obtains a distinct universal vertex. The extra universal vertex added to $G \odot K_{C_{\mathbf{ch}}}$ guarantees that the equivalence class with no neighbors in Z' is also covered. This completes the proof. □

Lemma 8.32. *Let (G, k) be an instance of* Connected Dominating Set*, where G is a connected graph of bounded expansion. Let $C_{\boldsymbol{S}} = (4C_{\boldsymbol{eq}} + 1) \cdot C_{\boldsymbol{eq}} \cdot 3^{C_{\boldsymbol{eq}}}$ and $c = (4C_{\boldsymbol{eq}} + 1)$. Then, for any fixed $\epsilon > 0$, there is a polynomial-time algorithm that either concludes that $\mathbf{cds}(G) > k$ or outputs a set $Y \subseteq V(G)$ of cardinality $\mathcal{O}(f(\epsilon) \cdot k)$, for some function f, and a set $Z \subseteq Y$, such that (i) Z is a c-exchange domination core in G and (ii) $OPT_{SCDS}((G[Y], Z), k) \leq (1 + \epsilon) OPT_{CDS}(G, k)$.*

Proof. We start by designing an algorithm $\mathcal{A}$ with the desired properties. Starting with Lemma 8.22, Algorithm $\mathcal{A}$ either concludes that $\mathbf{cds}(G) > k$ or finds (in polynomial time) a c-exchange domination core Z of size at most $\mathcal{O}(k)$. Since Z is a Z-dominator itself, using Proposition 8.21, we find a set O, such that $|O| \leq |Z|$ and O dominates G. Moreover, by Proposition 8.1, we find a connected superset $\ddot{O}$ of O, such that $|\ddot{O}| \leq 3|O|$. Next, we let $\dot{Z}$ be the set $Z \cup \ddot{O}$ and compute $Z' = \mathrm{cl}_1(\dot{Z})$ using Lemma 8.9; then we have that $|Z'| = \mathcal{O}(k)$. We then partition $V(G) \setminus Z'$ into equivalence classes with respect to $\simeq$ (i.e. $u \simeq v \quad \Leftrightarrow \quad M_1(u, Z') = M_1(v, Z')$). From Lemma 8.30, we know that the equivalence relation $\simeq$ has at most $\mathcal{O}(k)$ classes. Let X' be the set, again of size $\mathcal{O}(k)$, obtained after applying Lemma 8.27 to graph $\ddot{G}$ with $X = Z' \cup U$, $r = 2$, and $q = 2t$ (we fix the value of t later and U is the set defined earlier).

Since Z' dominates the c-exchange domination core Z, from Proposition 8.21 it follows that we can find a set $\ddot{Z}$, such that $|\ddot{Z}| \leq |Z'|$ and $\ddot{Z}$ dominates G. Hence, we can apply Proposition 8.1 to obtain a set W of size at most $4|Z'|$ such that $Z' \cup \ddot{Z} \cup W$ is a connected dominating set in G. Algorithm $\mathcal{A}$ outputs instance $((G', Z), k) = ((G[Y], Z), k)$ which asks to dominate Z, where $Y = Z' \cup \ddot{Z} \cup W \cup (X' \setminus U)$. Note that, since we remove the set U and add W, the

graph $G[Y]$ is a connected induced subgraph of the original graph G. Moreover, the total number of vertices in $G[Y]$ is at most $\mathcal{O}(k)$. The correctness of the algorithm follows from the correctness of each step.

We now show that $\mathrm{OPT}_{\mathrm{SCDS}}((G[Y], Z), k) \leq (1+\epsilon)\mathrm{OPT}_{\mathrm{CDS}}(G, k)$. Consider the graph D^* induced by an optimal connected dominating set of G (i.e. D^* is a tree). If $|V(D^*)| > k$, then $\mathrm{OPT}_{\mathrm{SCDS}}((G[Y], Z), k) \leq (1+\epsilon)\mathrm{OPT}_{\mathrm{CDS}}(G, k)$ holds trivially. So we assume that $|V(D^*)| \leq k$. We let $\mathcal{F}(D^*, t) = \{T_1, T_2, \cdots\}$ denote a (D^*, t)-covering family. Proposition 8.3 implies that $|\mathcal{F}(D^*, t)| \leq \frac{|V(D^*)|}{t} + 1$ and $\sum_{T \in \mathcal{F}(D^*,t)} |V(T)| \leq (1 + \frac{1}{t})|V(D^*)| + 1$. Moreover, the size of each subtree T is at most $2t$. We define groups on $V(G)$ as follows. Each vertex $v \in Z'$ belongs to a unique group q_v and each vertex in $V(G) \setminus Z'$ belongs to group q_κ, i.e. vertices in the same equivalence class in $\simeq$ belong to the same group.

Claim 8.33. *For any $T \in \mathcal{F}(D^*, t)$ there exists a tree T' in G' of size at most $|V(T)|$ which contains at least one vertex from each group appearing in T.*

Proof. Recall that, when constructing the graph $\ddot{G}$, we added one universal vertex for each equivalence class in $V(G) \setminus Z'$. Hence, after applying Lemma 8.27, we know that for any $Y \subseteq Z' \cup U$ of size at most $2t$ (which is exactly a subset of the groups) if $\mathbf{st}_G(Y) \leq 4t$ then $\mathbf{st}_{\ddot{G}}(Y) = \mathbf{st}_G(Y)$. Every vertex in Z' belongs to a distinct group and every vertex $u_\kappa \in U$ is connected to all vertices in κ. Hence, any tree of size greater than one containing a vertex u_κ must also contain a neighbor of u_κ (from the same group). The existence of T implies that there exists a tree of size at most $2t$ connecting all groups appearing in T. Hence, a tree T' certifying this fact exists in G'. □

We now construct a new family $\mathcal{F}'$ which consists of replacing each $T \in \mathcal{F}(D^*, t)$ by a set T' in G'. Let $D'' = \bigcup_{T' \in \mathcal{F}'} V(T')$. By the previous claim and the fact that $\sum_{T \in \mathcal{F}(D^*,t)} |V(T)| \leq (1+\frac{1}{t})|V(D^*)|+1$, we know that $|D''| \leq (1+\frac{1}{t})|V(D^*)|+1$ and D'' dominates Z' (since we never reduce the number of groups in D''). Moreover, D'' consists of at most $\frac{|V(D^*)|}{t}+1$ components. Note that $\ddot{O}$ is a connected Z-dominator and therefore the set $\ddot{O} \cup Z$ is also connected. Now, since $Z \cup \ddot{O} \subseteq Z'$, D'' dominates $Z \cup \ddot{O}$ and applying Proposition 8.1, we obtain a connected $(Z \cup \ddot{O})$-dominator, and hence also Z-dominator, D' of size at most $\frac{2|V(D^*)|}{t} + 2 + (1 + \frac{1}{t})|V(D^*)| + 1 = \frac{2|V(D^*)|}{t} + 3 + |V(D^*)| + \frac{|V(D^*)|}{t} = (1 + \frac{3}{t})|V(D^*)| + 3$. Setting t appropriately (i.e. roughly $\frac{3}{\epsilon}$) completes the proof. □

Using the same arguments as in the proof of Theorem 28 and replacing Lemma 8.18 by Lemma 8.32, we obtain the following:

Theorem 29. *For every $\epsilon > 0$, CDS admits a $(1+\epsilon)$-approximate bikernel with $\mathcal{O}(f(\epsilon) \cdot k)$ vertices on graphs of bounded expansion, where f is some computable function.*

Part III

Concluding remarks

Chapter 9

Conclusions and future directions

In the thesis, we have studied the Feedback Vertex Set problem in tournaments and bipartite tournaments (Chapters 5 and 6), a generalization of Vertex Cover known as ℓ-Component Order Connectivity (Chapter 7) and Connected Dominating Set (Chapter 8). Starting with a formulation of these problems as a Hitting Set problem, we can get algorithms and kernels (except for CDS) which we have improved significantly. Yet, a lot is left to be explored and we outline some of the future directions to take.

Feedback Vertex Set In chapter 5 and 6, we provided FPT algorithms with running time $1.6181^k \cdot n^{\mathcal{O}(1)}$. It would be interesting to improve these algorithms. We hope that the same template as in these chapters could be applicable in several situations where one seeks a "small" set of vertices or edges to delete in order to modify the input graph to a "rigid" structure; such as Cluster Vertex Deletion, Cograph Vertex Deletion and Feedback Vertex Set in the more general setting when the input graph is a multi-partite tournament [GY08].

ℓ-Component Order Connectivity In chapter 7, we provided a kernel for ℓ-component order connectivity of size $2\ell k$ which is tight when $\ell = 1$. Is this algorithm tight for general values of ℓ? We also provide a separation oracle for the problem that runs in $(3e)^\ell \cdot n^{\mathcal{O}(1)}$ time. A natural question is can this running time be improved? Note that by Set Cover Conjecture, we have that there is no algorithm that solves general Set Cover in time $(2-\epsilon)^n$ for for any $\epsilon > 0$ where n is the size of the universe. As the reduction used for the NP-completeness of Min ℓ-Connected Subgraph uses the Set Cover problem with $\ell = n + k$ (k being the solution size of set cover), we have that Min ℓ-Connected Subgraph can not be solved in $(2-\epsilon)^\ell$ time for any $\epsilon > 0$. Note that via personal communications with Mingyu Xiao, we came to know that he has a similar generalization of the Weighted Expansion Lemma and his algorithm outputs a kernel of size $9\ell k$ and runs in polynomial time. It would be interesting to know whether our kernel size can be achieved in polynomial time.

Lossy Kernelization Lossy kernelization is a very new framework and most of the problems are still open in this area. A good introductory source is the work of Lokshtanov et al [LPRS16]. In Chapter 8, we showed that Connected Dominating Set admits an α-approximate bikernel on graphs of bounded expansion and an α-approximate kernel on $K_{d,d}$-free graphs, for every $\alpha > 1$. For $K_{d,d}$-free graphs we obtain instances of size $k^{\mathcal{O}(\frac{d^2}{\alpha})}$ while for bounded expansion graphs we obtain instances of size $\mathcal{O}(f(\alpha)k)$ (i.e, linear in k), where $f(\alpha)$ is a computable function depending only on α.

Note that the size of our kernel for graphs of bounded expansion matches the size of the best known kernel for Dominating Set, while for $K_{d,d}$-free graphs we have an additional $\frac{1}{\epsilon}$ multiplicative factor in the exponent. This leads us to the following two interesting open questions. Is it possible to reduce the size of our kernel on $K_{d,d}$-free graphs to $f(\epsilon)k^{\mathcal{O}(d^2)}$ for some function f? And, in light of the $\mathcal{O}(k^{(d-1)(d-3)-\epsilon})$ lower bound for Dominating Set, it is possible to obtain a lossy kernel for Dominating Set on biclique-free graphs that beats this bound?

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[AFN04] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[AG08] N. Alon and S. Gutner. Kernels for the dominating set problem on graphs with an excluded minor. *Electronic Colloquium on Computational Complexity*, 15(066), 2008.

[AK10] F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, November 2010.

[AMS06] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, April 2006.

[AP14] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. In *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 271, 2014.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the Association for Computing Machinery*, 42(4):844–856, 1995.

[BBF99] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.

[BD05] I. Bezáková and V. Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

[BDFH09] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[BEF+06] K. Burrage, V. Estivill-Castro, M. R. Fellows, M. A. Langston, S. Mac, and F. A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 192–202, 2006.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[BFL+09] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, Atlanta, Georgia, USA*, pages 629–638, 2009.

[BG95] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[BHKK07] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, pages 67–74, New York, NY, USA, 2007.

[BKMN16] K. Bringmann, L. Kozma, S. Moran, and N. S. Narayanaswamy. Hitting set for hypergraphs of low VC-dimension. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 23:1–23:18, 2016.

[BL82] L. W. Beineke and C. H. Little. Cycles in bipartite tournaments. *Journal of Combinatorial Theory, Series B*, 32(2):140 – 145, 1982.

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[BvD10] H. L. Bodlaender and T. C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

[Cai]

[Cai96] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171 – 176, 1996.

[CC04] M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems. In *Algorithms - ESA 2004, 12th Annual European*

Symposium, Bergen, Norway, September 14-17, 2004, Proceedings, pages 192–203, 2004.

[CC15] L. Cai and Y. Cai. Incompressibility of h-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015.

[CCH+16] M. Chang, L. Chen, L. Hung, P. Rossmanith, and P. Su. Fixed-parameter algorithms for vertex cover p3. *Discrete Optimization*, 19:12–22, 2016.

[CCL10] Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 93–104, 2010.

[CDL+16] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41, 2016.

[CDZ00] M. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2000.

[CFK+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CFL+08] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

[CGH13] M. Cygan, F. Grandoni, and D. Hermelin. Tight kernel bounds for problems on graphs with small degeneracy - (extended abstract). In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 361–372, 2013.

[CKJ01] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[CKX10] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

[Cla85] C. Clapham. The bipartite tournament associated with a fabric. *Discrete Mathematics*, 57(1):195 – 197, 1985.

[CLL+08] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008.

[CNP+11] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.

[CP12] M. Cygan and M. Pilipczuk. On fixed-parameter algorithms for split vertex deletion. *CoRR*, abs/1208.1248, 2012.

[CPPW12] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. *Discrete Applied Mathematics*, 160(15):2131–2141, 2012.

[DDF+16] P. G. Drange, M. S. Dregi, F. V. Fomin, S. Kreutzer, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, F. Reidl, F. S. Villaamil, S. Saurabh, S. Siebertz, and S. Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 31:1–31:14, 2016.

[DDLS15] P. G. Drange, M. S. Dregi, D. Lokshtanov, and B. D. Sullivan. On the threshold of intractability. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 411–423, 2015.

[DDvtH14] P. G. Drange, M. S. Dregi, and P. van 't Hof. On the computational complexity of vertex integrity and component order connectivity. In *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, pages 285–297, 2014.

[DF95] R. G. Downey and M. R. Fellows. *Parameterized Computational Feasibility*, pages 219–244. Birkhäuser Boston, Boston, MA, 1995.

[DF97] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[DFL+05] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $\mathcal{O}(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, pages 859–869, 2005.

[DGH+06] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, pages 320–331, 2006.

[DGH+10] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

[Die12] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DM12] H. Dell and D. Marx. Kernelization of packing problems. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 68–81, 2012.

[Dru15] A. Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

[DS05] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[DvM14a] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

[DvM14b] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

[EG04] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

[ENSS98] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[ERS05] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the vc-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005.

[Fei98] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[FG06] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

[FGK09] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009.

[FGK+10] F. V. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, and S. Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010.

[FGLS16] F. V. Fomin, S. Gaspers, D. Lokshtanov, and S. Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 764–775, 2016.

[FGPR08] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

[FH66] L. M. Frank Harary. The theory of round robin tournaments. *The American Mathematical Monthly*, 73(3):231–246, 1966.

[FK10] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[FKW04] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Proceedings of the 30th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 245–256, Berlin, Heidelberg, 2004. Springer-Verlag.

[FLST10] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510, 2010.

[FLST12] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 82–93, 2012.

[FLST13] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 92–103, 2013.

[FS11] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

[GGH+06] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

[GHI+13] D. Gross, M. Heinig, L. Iswara, W. Kazmierczak, K. Luttrell, J. T. Saccoman, and C. Suffel. A survey of component order connectivity models of graph theoretic networks. 12:895–910, 2013.

[GHM+11] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011.

[GJ79a] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJ79b] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman and Co., 1979.

[GK98] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[GKW08] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *Comput. J.*, 51(3):372–384, 2008.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics. Springer-Verlag, Berlin, New York, 1988.

[GM13] S. Gaspers and M. Mnich. Feedback vertex sets in tournaments. *Journal of Graph Theory*, 72(1):72–89, 2013.

[GPP10] S. Guillemot, C. Paul, and A. Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. In *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, pages 147–157, 2010.

[GY08] G. Gutin and A. Yeo. Some parameterized problems on digraphs. *Comput. J.*, 51(3):363–371, 2008.

[HKL+13] P. Heggernes, D. Kratsch, D. Lokshtanov, V. Raman, and S. Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Inf. Comput.*, 231:109–116, 2013.

[HKMN10] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47(1):196–217, 2010.

[HL12] S. Har-Peled and M. Lee. Weighted geometric set cover problems revisited. *JoCG*, 3(1):65–85, 2012.

[HMvLW11] D. Hermelin, M. Mnich, E. J. van Leeuwen, and G. J. Woeginger. Domination when the stars are out. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 462–473, 2011.

[Hoc82] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982.

[HS81] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[Hsi11] S. Hsiao. Fixed-parameter complexity of feedback vertex set in bipartite tournaments. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 344–353, 2011.

[HW12] D. Hermelin and X. Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 104–113, 2012.

[ID05] S. S. Irit Dinur. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[IMR+98] H. B. H. III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Nc-approximation schemes for NP- and pspace-hard problems for geometric graphs. *J. Algorithms*, 26(2):238–274, 1998.

[IP99] R. Impagliazzo and R. Paturi. The complexity of k-sat. In *COCO '99: Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, page 237, Washington, DC, USA, 1999. IEEE Computer Society.

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[KD79] M. S. Krishnamoorthy and N. Deo. Node-deletion np-complete problems. *SIAM J. Comput.*, 8(4):619–625, 1979.

[Kho02] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775, 2002.

[KKS11] F. Kardos, J. Katrenic, and I. Schiermeyer. On computing the minimum 3-path vertex cover and dissociation number of graphs. *Theor. Comput. Sci.*, 412(50):7009–7017, 2011.

[KL16a] M. Kumar and D. Lokshtanov. A 2lk kernel for l-component order connectivity. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, pages 20:1–20:14, 2016.

[KL16b] M. Kumar and D. Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in bipartite tournaments. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 24:1–24:15, 2016.

[KL16c] M. Kumar and D. Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 49:1–49:13, 2016.

[KM86] M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.

[Kom15] C. Komusiewicz. Tight running time lower bounds for vertex deletion problems. *CoRR*, abs/1511.05449, 2015.

[KP14] T. Kociumaka and M. Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

[KR02] S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289(2):997–1008, 2002.

[KR08] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[Kra14] S. Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

[KW13] S. Kratsch and M. Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013.

[LMP+15] D. Lokshtanov, A. E. Mouawad, F. Panolan, M. S. Ramanujan, and S. Saurabh. Reconfiguration on sparse graphs. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 506–517, 2015.

[LMS11] D. Lokshtanov, M. Mnich, and S. Saurabh. A linear kernel for a planar connected dominating set. *Theor. Comput. Sci.*, 412(23):2536–2543, 2011.

[LMS12] D. Lokshtanov, N. Misra, and S. Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.

[LPRS16] D. Lokshtanov, F. Panolan, M. S. Ramanujan, and S. Saurabh. Lossy kernelization. *CoRR*, abs/1604.04111, 2016.

[LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

[LY80a] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

[LY80b] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994.

[MMS15] S. M. Meesum, P. Misra, and S. Saurabh. Reducing rank of the adjacency matrix by graph modification. In *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, pages 361–373, 2015.

[Mos15] D. Moshkovitz. The projection games conjecture and the NP-hardness of $\ln n$-approximating set-cover. *Theory of Computing*, 11:221–235, 2015.

[MPR+10] N. Misra, G. Philip, V. Raman, S. Saurabh, and S. Sikdar. FPT algorithms for connected feedback vertex set. In *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings*, pages 269–280, 2010.

[MS16] S. M. Meesum and S. Saurabh. Rank reduction of directed graphs by vertex and edge deletions. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 619–633, 2016.

[MWV16] M. Mnich, V. V. Williams, and L. A. Végh. A 7/3-approximation for feedback vertex sets in tournaments. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 67:1–67:14, 2016.

[NdM10] J. Nesetril and P. O. de Mendez. From sparse graphs to nowhere dense structures: Decompositions, independence, dualities and limits, 2010. European Congress of Mathematics.

[NG10] J. Nastos and Y. Gao. Bounded search tree algorithms for parameterized cograph deletion: Efficient branching rules by exploiting structures of special graph classes. *CoRR*, abs/1006.3020, 2010.

[NHK08] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4), 2008.

[NJ74] G. L. Nemhauser and L. E. T. Jr. Properties of vertex packing and independence system polyhedra. *Math. Program.*, 6(1):48–61, 1974.

[NR03] R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003.

[NSS95] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 182–191, 1995.

[PM81] A. Paz and S. Moran. Non deterministic polynomial optimization problems and their approximations. *Theor. Comput. Sci.*, 15:251–277, 1981.

[PRS09] G. Philip, V. Raman, and S. Sikdar. Polynomial kernels for dominating set in $K_{i,j}$-free and d-degenerate graphs. *CoRR*, abs/0903.4521, 2009.

[PRS12] G. Philip, V. Raman, and S. Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11, 2012.

[PY96] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.*, 53(2):161–170, 1996.

[Raz06] I. Razgon. Exact computation of maximum induced forest. In *Algorithm Theory - SWAT 2006, 10th ScandinavianWorkshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*, pages 160–171, 2006.

[Raz07] I. Razgon. Computing minimum directed feedback vertex set in $o(1.9977^n)$. In *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81, 2007.

[Rob86] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[RRST96] B. Reed, N. Robertson, P. Seymour, and R. Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, New York, NY, USA, 1997. ACM.

[RS06] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci.*, 351(3):446–458, 2006.

[RS08] V. Raman and S. 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.

[RSS06] V. Raman, S. Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Trans. Algorithms*, 2(3):403–415, 2006.

[Sas08] P. Sasatte. Improved FPT algorithm for feedback vertex set problem in bipartite tournament. *Inf. Process. Lett.*, 105(3):79–82, 2008.

[Sey95] P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

[Spe89] E. Speckenmeyer. On feedback problems in diagraphs. In *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89, Castle Rolduc, The Netherlands, June 14-16, 1989, Proceedings*, pages 218–231, 1989.

[Tho10] S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), 2010.

[Tru05] A. Truß. Parameterized algorithms for feedback set problems in tournaments. Master's thesis, Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2005.

[Tu15] J. Tu. A fixed-parameter algorithm for the vertex cover p_3 problem. *Inf. Process. Lett.*, 115(2):96–99, 2015.

[TV12] J. A. Telle and Y. Villanger. FPT algorithms for domination in biclique-free graphs. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 802–812, 2012.

[TZ11] J. Tu and W. Zhou. A factor 2 approximation algorithm for the vertex cover p3 problem. *Inf. Process. Lett.*, 111(14):683–686, July 2011.

[Vaz03] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2):264–280, 1971.

[Wah07] M. Wahlström. Algorithms, measures and upper bounds for satisfiability and related problems. Technical report, Department of Computer, 2007.

[WS11] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[XK15] M. Xiao and S. Kou. Faster computation of the maximum dissociation set and minimum 3-path vertex cover in graphs. In *Frontiers in Algorithmics - 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings*, pages 282–293, 2015.

[XN13] M. Xiao and H. Nagamochi. Exact algorithms for maximum independent set. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 328–338, 2013.

[XN15] M. Xiao and H. Nagamochi. An improved exact algorithm for undirected feedback vertex set. *J. Comb. Optim.*, 30(2):214–241, 2015.

[Yan81] M. Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.