Parameterized Graph Separation Problems: New Techniques and Algorithms

By

M. S. Ramanujan

MATH10200905002

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

For the Degree of

DOCTOR OF PHILOSOPHY
of
HOMI BHABHA NATIONAL INSTITUTE

July, 2013

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by M. S. Ramanujan entitled Parameterized Graph Separation Problems: New Techniques and Algorithms and recommend that it may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

____ Date:
Chair : V. Arvind

____ Date:
Guide/Convener : Saket Saurabh

____ Date:
Co-Guide/Member 1 : Venkatesh Raman

____ Date:
Member 2 : C. R. Subramanian

____ Date:
Member 3 : Raghavendra Rao B. V.

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

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

Date:

Place: Guide

STATEMENT BY AUTHOR

DECLARATION

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

M. S. Ramanujan

DEDICATIONS

This thesis is dedicated to my sister Rohini and my parents for all the love and support I have received from them in every one of my endeavors.

ACKNOWLEDGEMENTS

I am deeply indebted to my thesis advisor Professor Saket Saurabh and co-advisor Professor Venkatesh Raman for their invaluable guidance which made this work possible.

I am extremely grateful to Professors V. Arvind, Kamal Lodaya, C. R. Subramaniam, R. Ramanujam, Meena Mahajan and Vikram Sharma for granting me an opportunity to learn from their vast knowledge and experience.

I will be forever grateful to Professor K. Muralikrishnan for his guidance during my undergraduate days, for inspiring me to take up research and for introducing me to the exciting world of Parameterized Complexity.

I am grateful to Professors Fedor Fomin, Gregory Gutin and Fabrizio Grandoni for their kind invitations to collaborate with their respective groups and for extremely enriching research visits. I would like to acknowledge the sources of funding for these visits, an NFR project on Network Searching, a Royal Society International (Indo-UK) Joint Project and a DST Indo-Swiss project. I would also like to acknowledge the funding from the Institute of Mathematical Sciences which has enabled me to travel and present my papers at various conferences.

I would like to thank Geevarghese Philip, Neeldhara Misra, Daniel Lokshtanov and Somnath Sikdar from whom I have learnt a lot during our discussions on specific problems as well as about research in general. I would also like to thank Pim Van't Hof, Michal Pilipczuk, Mark Jones, Robert Crowston and Gabriele Mucciacia for making my research visits to Bergen and London extremely enjoyable. I am grateful to Manu Basavaraju for many an enlightening discussion as well as for the table tennis lessons. Special thanks go to R. Kavya for all the encouragement and motivation during the writing of this thesis. I am grateful to Aishwarya Cyriac, Yadu Vasudev, Karteek Sreenivasaiah, Nitin Saurabh, Pranabendu Misra, Mrinal Kumar, Sreejith A. V., M. Praveen, Kunal Dutta, Mathew C. Francis, Gaurav Rattan, Fahad Panolan, Ashutosh Rai, Sudeshna Kolay, Syed Mohammed Meesum for always being open to discussions in matters relating to science or otherwise.

I am extremely grateful to R. Sriraman and Gayatri Ajay for all the excellent advice they have always given me and for being such inspiring figures. Finally, thank you Swathi Govindarajan for all the pizzas!

Contents

Synopsis

A fundamental theorem about connectivity in finite undirected graphs is Menger's Theorem [60], which states the following.

> *Let G be a finite undirected graph and s and t be two nonadjacent vertices. Then, the minimum number of vertices whose removal disconnects s and t is equal to the maximum number of pairwise vertex disjoint paths from s to t.*

There is a corresponding edge version of this theorem and also vertex and arc versions on directed graphs. This *min-max* theorem is an extremely fundamental one in combinatorial optimization and the minimum s-t cut can be computed in polynomial time [1]. In this thesis, we refer to generalizations of the problem of finding the minimum set of vertices disconnecting two vertices in graph, as graph separation problems. Some natural examples include the problem of finding the minimum number of vertices disconnecting vertices of a given set from each other, and the problem of finding the minimum number of vertices disconnecting multiple specified pairs of vertices.

It turns out that these problems are computationally intractable in the classical setting, and therein lies the motivation behind the study of these problems from the point of view of the field of Parameterized Complexity.

The goal of Parameterized Complexity is to find ways of solving NP-hard problems more efficiently than by brute force. Here, the aim is to restrict the combinatorial explosion of computational difficulty of the problems to a parameter that is hopefully much smaller than the input size. An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is a non-negative integer. A parameterized problem is *fixed-parameter tractable* if there exists a computable function

f and a constant c such that instances (I, k) can be solved in time $O(f(k)\|I\|^c)$ where $\|I\|$ denotes the size of I. FPT is the class of all fixed-parameter tractable problems and algorithms which run in the time specified above are called FPT algorithms.

Marx [73] initiated the study of graph separation problems in Parameterized Complexity by studying the MULTIWAY CUT problem where the input is an undirected graph, a set of terminals and a budget k and the objective will be to test if there are k vertices which intersect all paths between every pair of terminals. This problem is known to be NP-hard [25] even for 3 terminals. Marx gave the first FPT algorithm for this problem which did not rely on results from Graph Minors. In the course of giving his FPT algorithm, he introduced a notion of "important separators" which has subsequently proved to be an extremely fundamental tool in obtaining FPT algorithms for a large number of graph separation problems and consequently settling many open problems. Notable examples include the first FPT algorithms for DIRECTED FEEDBACK VERTEX SET [17], ALMOST 2-SAT [90], and MULTICUT [79, 10].

In recent years, the area of design of parameterized algorithms for graph separation problems has seen a lot of activity resulting in new algorithms and techniques. In a parallel line of research, a lot of classical problems which are not graph separation problems or even graph problems in some cases have been found to have a graph separation problem at their core. Notable examples of this include the VERTEX COVER problem [88], the ALMOST 2-SAT [90] problem, DELETION q-Horn BACKDOOR SET DETECTION [38], and SATISFIABILITY [38]. The frequent occurrences of graph separation problems at the heart of various seemingly unrelated problems has also motivated the study of these problems and the search for improved algorithms and more powerful and general techniques.

In this thesis, we will present results extending in both the directions mentioned above. That is, we

- design new techniques and frameworks to obtain new as well as improved FPT algorithms for certain kinds of parameterized graph separation problems;
- present problems which are not graph separation problems themselves, but have some variant of graph separation at their core, after which by using our new frameworks as well as existing ones, we give new as well as improved FPT algorithms.

In particular, we first study the VERTEX COVER problem and give improved FPT algorithms with two different parameters. The first algorithm is parameterized by the size of the minimum vertex cover above the size of a maximum matching. We first prove structural characterizations relating maximum matchings and minimum vertex covers which extend the known results in this direction. Following this, we show that this structural characterization in fact proves that we can cast the VERTEX COVER problem as a graph separation problem and by applying the notion of important separators, we improve upon the previous best algorithm.

The second algorithm for VERTEX COVER is parameterized by the size of the minimum vertex cover above the size of the optimum solution to the standard LP relaxation of the Vertex Cover Integer Linear Program. This algorithm is an example of the power of the recently developed framework of parameterizing above LP and as a consequence of this algorithm, we get the first improved FPT algorithm for the ODD CYCLE TRANSVERSAL problem after the initial algorithm given in 2003. This result also improves upon the previous best bounds for a number of problems, all of whom have VERTEX COVER with this particular parameter as a generalization.

We also introduce a generalization of the concept of important separators which is applicable to many problems where important separators break down. Furthermore, we also design a generic subroutine which imposes certain structure on a given graph, which can then be exploited to design FPT algorithms for certain problems. We demonstrate the power of this technique by introducing a parity based generalization of the MULTIWAY CUT problem, called PMWC. Parity based graph separation problems have attracted a lot of attention in recent years and we add to the results and techniques in this direction with our FPT algorithm for PMWC and the generalization of important separators.

The next framework we introduce is an application of important separators to obtain greedy approximation algorithms for graph separation problems. Following this, we consider the DELETION q-Horn BACKDOOR SET DETECTION and SATISFIABILITY problems and show that both of these problems can be recast as graph separation problems and by applying our framework in spite of it being designed only for graph separation problems, obtain FPT-approximation algorithms for the above two problems as well. This gives us the first FPT-approximation algorithm for DELETION q-Horn BACKDOOR SET DETECTION as well as the first FPT algorithm for SATISFIABILITY parameterized by the size of the smallest q-Horn de letion set.

Finally, we give a framework for obtaining *efficient* FPT algorithms for a number of problems. Here, *efficient* implies that the dependence of the running time on the input size is linear. We introduce a graph separation problem on skew-symmetric graphs, called the d-SKEW-SYMMETRIC MULTICUT problem and give an FPT algorithm for this problem which has a linear dependence on the input size. Following this, we show that a large number of problems studied in parameterized complexity have parameter preserving linear time reductions to this problem and therefore, obtain linear time FPT algorithms for all these problems for which the existence of such algorithms was an open problem over the last decade. The most notable example is the ODD CYCLE TRANSVERSAL problem, which has been known to have linear time FPT algorithms only on special graph classes and for which the existence of a linear time FPT algorithm on general graphs was an open question since 2003. However, using the linear time parameter preserving reduction from this problem to d-SKEW-SYMMETRIC MULTICUT, we answer this question in the affirmative.

As a result of our techniques and frameworks we get the following new results.

New FPT algorithms

1. We obtain a $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$ algorithm for VERTEX COVER parameterized above the size of the optimum value of the LP where n is number of vertices in the input graph.

2. We introduce a parity based generalization of the classical MULTIWAY CUT problem, called the PARITY MULTIWAY CUT problem and give an algorithm for this problem which runs in time $\mathcal{O}(2^{\mathcal{O}(k^3)} n^{\mathcal{O}(1)})$, where n is the number of vertices in the input graph.

3. We introduce a graph separation problem on skew-symmetric graphs, namely the d-SKEW-SYMMETRIC MULTICUT problem, show that it generalizes numerous well studied parameterized problems and give an algorithm which runs in time $\mathcal{O}((4d)^k k^4 (m+n))$ where m and n are the number of edges and vertices in the input graph respectively.

4. We obtain an FPT algorithm for DELETION q-Horn BACKDOOR SET DETECTION which runs in time $\mathcal{O}(12^k k^5 \ell)$ where ℓ is the length of the input formula.

5. We obtain an FPT algorithm for SATISFIABILITY which runs in time $\mathcal{O}(12^k k^5 \ell)$ where k is the size of the smallest q-Horn deletion backdoor set of the given formula and ℓ is the length of the input formula.

FPT algorithms with improved dependence on the parameter

1. We obtain $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$ algorithms for VERTEX COVER parameterized above the size of the maximum matching and ODD CYCLE TRANSVERSAL where n is the number of vertices in the input graph. This is the first improvement (with respect to the parameter) for ODD CYCLE TRANSVERSAL after the first algorithm of Reed et al. (2004).

2. We obtain $\mathcal{O}(2.3146^k \ell^{\mathcal{O}(1)})$ algorithms for ALMOST 2-SAT where ℓ is the length of the input formula.

FPT algorithms with improved dependence on the input size

1. We obtain an algorithm for ODD CYCLE TRANSVERSAL which runs in time $\mathcal{O}(4^k k^4(m+n))$, which is the first linear time FPT algorithm for this problem and answers a question asked by Reed et al. (2004).

2. We obtain an algorithm for ALMOST 2-SAT which runs in time $\mathcal{O}(4^k k^4 \ell)$ and an algorithm for VERTEX COVER parameterized above the size of the maximum matching running in time $\mathcal{O}(4^k k^4(m+n))$ given a graph with a matching.

Organization of Thesis. Chapters 1 and 2 contain description of basic notations and definitions regarding Parameterized Complexity. Chapter 3 gives the intuition behind the notion of important separators, followed by the formal definitions of both vertex and arc separators, and finally concluding with the presentation of a template using which we will describe some of the algorithms which appear later in the thesis. In Chapter 4, we recall some definitions and classical results regarding matchings and vertex covers and using these, prove a structural characterization which extends the well known König-Egerváry theorem. In Chapter 5, we use the characterization proved in the previous chapter along with the template for important separators given in Chapter 3 to give an improved FPT algorithm for the ABOVE GUARANTEE VERTEX COVER problem. In Chapter 6, we consider the parameterization of the VERTEX COVER problem above the value of the optimum LP solution and give the first FPT algorithm for this problem and as a consequence obtain improved FPT algorithms for a number of parameterized problems. In Chapter 7, we introduce a generalization of important separators and use it to give an FPT algorithm for the PMWC problem. In Chapter 8, we introduce a framework of applying important separators to obtain greedy polynomial time approximation algorithms and illustrate it with the MULTIWAY CUT problem as an example. In Chapter 9, we study the problem of computing the smallest backdoor sets to the class of q-Horn formulas and apply the framework described in the previous chapter to obtain an FPT algorithm for DELETION q-Horn BACKDOOR SET DETECTION. In Chapter 10, we develop a framework to obtain linear time FPT algorithms for a number of problems by introducing the d-SKEW-SYMMETRIC MULTICUT problem and giving the first FPT algorithm for this problem which also has a linear dependence on the input size.

List of Figures

Part I

Introduction

1

Parameterized Complexity

In this Chapter we give a broad overview of the field of Parameterized Complexity.

1.1 Introduction

Definition 1.1.1. ([85]) *A* parameterized problem *is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the* parameter *of the problem.*

Definition 1.1.2. ([85]) *A parameterized problem L is said to be* Fixed Parameter Tractable *(FPT) if it can be determined in time $f(k) \cdot n^{\mathcal{O}(1)}$ whether or not $(x, k) \in L$, where f is a computable function depending only on k and $n = |(x, k)|$. The complexity class containing all fixed parameter tractable problems is called FPT.*

Unless specified otherwise, we will assume that the parameter k is a non negative integer encoded with a unary alphabet. In the thesis, the parameterized problems will be either on graphs or on CNF-formulas and the objective will to test if there is a set of k vertices/edges or variables/clauses which satisfy a certain property. We refer to such sets as *solutions* to the given instance.

Definition 1.1.3. *Let $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized (many one) reduction from L_1 is a triple (f, f_1, f_2) where $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ and $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$ such that*

1. *$f(x, k)$ is computable in time $f_1(k).|(x, k)|^{\mathcal{O}(1)}$ and*

2. *$(x, k) \in L_1$ if and only if $(f(x, k), f_2(k)) \in L_2$.*

The parameterized reduction (f, f_1, f_2) is said to be a parameter preserving parameterized reduction if $f_2(k) = k$.

The *Weft Hierarchy* consists of parameterized complexity classes W[1] $\subseteq$ W[2] $\subseteq \cdots$ which are defined as the closure of certain parameterized problems under FPT-reductions (see [27, 33] for definitions). There is strong theoretical evidence that parameterized problems that are hard for classes W[i] are not fixed-parameter tractable. For example FPT $=$ W[1] implies that the Exponential Time Hypothesis (ETH) fails; that is, FPT $=$ W[1] implies the existence of a $2^{o(n)}$ algorithm for n-variable 3-SAT [33].

1.2 Some Standard Techniques for Designing FPT Algorithms

In this section, we give a summary of the standard FPT techniques used or referred to in this thesis.

1.2.1 Bounded Search Trees

The method of bounded search trees is one of the earliest known and simplest techniques used in the design of FPT algorithms. The main idea behind this technique is to find in polynomial time, a *small* set of elements such that at least one of them must occur in any feasible solution to the problem. Then we guess an element of this set to be in the solution we are trying to construct, process the input instance so that it reflects our choice of this element, make the necessary changes to the parameter value, and recursively solve the problem on the resulting instance. The small set of elements identified is usually a forbidden structure which needs to be removed and is usually of constant size, although the size of this set can be as large as logarithmic in the size of the input instance. As long as the depth of the recursion tree is bounded by some function of the parameter, since the time taken at each node of the recursion tree is polynomial, this results in an FPT algorithm. There is an easy $\mathcal{O}^*(2^k)$ algorithm for VERTEX COVER [80, 27] using this method.

1.2.2 Iterative Compression

Iterative Compression is a useful technique for designing FPT algorithms for minimization problems. This technique was first introduced in [91] to solve the ODD CYCLE TRANSVERSAL problem, where we are interested in finding a set of at most k vertices such that after removing these vertices, the resulting graph is bipartite. This method was also used in obtaining FPT algorithms for EDGE BIPARTIZATION, CHORDAL DELETION, CLUSTER VERTEX DELETION and FVS on undirected graphs [45, 75, 50, 26, 13]. This technique was also used by Chen et al. [17] to show that the DFVS problem is FPT, and by Razgon and Barry O'Sullivan [90] to show that ALMOST 2-SAT is FPT, two long standing open problems in the area of Parameterized Complexity. This technique is also used to design exact exponential algorithms (see [34]) and for other results based on this method, we refer the reader to a survey articles [47, 51].

We will give a sketch of this method as applied to a graph problem. The central idea here is to design an FPT algorithm which, when given a $k+1-$sized solution for a problem, either compresses it to a solution of size at most k or proves that there is no solution of size at most k. This is known as the compression step of the algorithm. The method adopted usually is to begin with a subgraph that trivially admits a $(k+1)-$sized solution and then expand it iteratively. In any iteration, we try find a compressed ($k-$sized) solution for the instance corresponding to the current subgraph. If we find such a solution, we use this solution and (usually) the vertex or edge we add to get the next subgraph, to get a $(k+1)-$sized solution for this instance and start the next iteration. We stop when we either get a solution of size k for the entire graph, or if some intermediate instance turns out to be incompressible. In order to stop when some intermediate instance turns out to be incompressible, the problem must have the property that the solution size in any subgraph is at most the solution size in the whole graph.

1.2.3 Kernelization

Kernelization is a polynomial time algorithm that preprocesses instances of problems and returns equivalent instances with a guaranteed upper bound on the size of the output, which is called the kernel of the instance. Kernelization is usually achieved by applying a set of reduction rules that allow us to remove or ignore parts of the instance that are easy to handle.

Kernelization has received increasing interest over the last decade, maturing from a technique to prove fixed parameter tractability, into a stand alone field of research. In recent years there has been many kernelization results for a variety of problems. Some notable examples are the linear vertex kernel for VERTEX COVER by Chen et al. [14], the quadratic vertex kernel for FVS in undirected graphs by Thomasse [95], and a polynomial kernel for MULTICUT in trees due to Bousquet et al. [11]. For further results we refer the reader to the survey articles [48, 6].

2

Notations, Definitions and Conventions

2.1 Basic Notations

We assume that the reader is familiar with basic notions like sets, functions, polynomials, relations, integers etc. In particular, for these notions we follow the same notations as that in [85].

2.2 Growth of Functions

We employ mainly the big-Oh ($\mathcal{O}$) notation (see [21]) and the big-Oh-star ($\mathcal{O}^*$) notation introduced in [100]. Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be two functions from Natural numbers to Natural numbers. We say that $f(n) = \mathcal{O}(g(n))$ if there exist constants c, and n_0 such that for all $n \geq n_0$, $f(n) \leq c.g(n)$. The notation $\mathcal{O}^*$ notation is essentially the big-Oh notation which hides polynomial factors and hence is used only for exponential time algorithms. We use $\mathcal{O}^*(f(n))$ to denote $\mathcal{O}(f(n).n^c)$ where c is some constant. In this thesis, we will use the $\mathcal{O}^*$ notation to hide factors polynomial in input size in order to focus on the function of the parameter. Hence, for us, $\mathcal{O}^*(f(k))$ denotes $\mathcal{O}(f(k).n^c)$ where k is the value of some parameter, n is the size of the input instance, and c is some constant.

2.3 Graphs

2.3.1 Undirected Graphs

An *undirected* graph G is a pair (V, E) where V and E are unordered sets. The elements of V are called vertices of G. E consists of unordered pairs of vertices and elements of E are called edges of G. A vertex u and a vertex v are said to be adjacent if E contains the pair (u, v). The edge (u, v) is said to be incident on the vertices u and v, while u and v are called the endpoints of the edge (u, v). An undirected graph G is called a *simple undirected* graph if there is no edge in E of the form (v, v) where v is a vertex of G. In this thesis, unless explicitly mentioned otherwise, the graphs we consider are all simple undirected graphs. The open neighborhood or just neighborhood of a vertex v in the graph G is the set $N(v) = \{u|(u, v) \in E\}$ and by closed neighborhood of a vertex v, we mean the set $N[v] = \{v\} \cup N(v)$. Let S be a set of vertices of G. We denote by (open) neighborhood of S, the set $N(S) = (\bigcup_{v\in S} N(v)) \setminus S$ and by closed neighborhood of S we denote the set $N(S) \cup S$.

A *walk* in the graph G is a sequence $W = v_1, \ldots, v_t$ of vertices such that $(v_i, v_{i+1}) \in E$ for every $1 \leq i \leq t-1$ and it is called a walk from v_i to v_t in G. The *length* of this walk is $t-1$. A walk in which any vertex occurs at most once is called a *path*. A walk where the first vertex is same as the last vertex and all the other vertices are distinct is called a *cycle*. The walks $v_i, v_{i+1}, \ldots, v_j$, $1 \leq i \leq t$, $i \leq j \leq t$ are called subwalks of the walk W. If W is a path these walks are called subpaths of W. A graph is called acyclic if it does not contain a cycle.

A vertex u is said to be *reachable* from a vertex v in G, if there is a path from u to v in G. The graph G is said to be *connected* if there is a path between every pair of vertices in G.

A set S of vertices (or edges) with some property is said to be a maximal set with that property if there is no set S' of vertices (respectively edges) such that $S' \supset S$ and S' has the same property.

A set S of vertices (or edges) with some property is said to be a minimal set with that property if there is no set S' of vertices (respectively edges) such that $S' \subset S$ and S' has the same property.

A connected component of G is an induced subgraph $X = G[C]$ of G such that C is a maximal subset of V such that $G[C]$ is connected.

A connected acyclic graph is called a tree and a graph whose connected components are trees is called a forest.

The line graph $L(G)$ of G is a graph where there is a vertex for every edge of G and two vertices of $L(G)$ are adjacent if and only if their corresponding edges are adjacent in G.

For a set T of vertices of G, we say that a path is a T-path if both endpoints of the path are in T and the internal vertices are disjoint from T.

2.3.2 Matchings

Given a graph $G = (V, E)$, a matching M in G is a set of pairwise non-adjacent edges. A vertex is called *matched* (or *saturated*) if it is an endpoint of one of the edges in the matching. Otherwise the vertex is *unmatched* or *unsaturated*. A vertex $v \in V$ is said to be *saturated* by M if it is the endpoint of an edge in M and it is *left unsaturated* by

M if v is not the endpoint of any edge in M. A maximum matching of the graph is a matching that contains the largest possible number of edges. A perfect matching is a matching which saturates every vertex of the graph. A set S of vertices is called a *vertex cover* of a given graph if every edge of the graph has an endpoint in the set S. The smallest such set of vertices is called the *minimum vertex cover* of the graph. Given a graph G, we use $\mu(G)$ and $\beta(G)$ to denote, respectively, the size of a maximum matching and a minimum vertex cover. A graph $G = (V, E)$ is said to be *König* if $\beta(G) = \mu(G)$.

2.3.3 Directed Graphs

A directed graph (or digraph) D is a pair (V, A) where V and A are sets. The elements of V are called vertices of D. The set A consists of ordered pairs of vertices and elements of A are called arcs of D. The arc (u, v) is said to be incident on the vertices u and v, while u and v are called the end points of the arc (u, v).

Furthermore, for an arc $a = (u, v) \in A$, we refer to u as the *tail* of this arc and denote it by **Tail**(a) and we refer to v as the *head* of this arc and denote it by **Head**(a). For a set of arcs P, we denote by **Tail**(P), the set $\bigcup_{a \in P}\{\textbf{Tail}(a)\}$ and we denote by **Head**(P) the set $\bigcup_{a \in P}\{\textbf{Head}(a)\}$. For a set of vertices V', we let $A[V']$ denote the set of arcs with both end points in the set V'. For a set of vertices V', we let $\delta^+(V')$ denote the set of arcs which have their tail in V' and their head in $V \setminus V'$. Similarly, we let $\delta^-(V')$ denote the set of arcs which have their head in V' and their tail in $V \setminus V'$. We also use $N^+(V')$ to denote the set **Head**$(\delta^+(V'))$ and $N^-(V')$ to denote the set **Tail**$(\delta^-(V'))$.

The undirected graph obtained from D by ignoring the ordering among the pairs constituting the arcs, is called the *underlying* undirected graph of D. A directed graph D is called a *simple directed* graph if there is no edge in A of the form (v, v) where v is a vertex of D. The out neighborhood of a vertex u in the digraph D is the set

$N^+(u) = \{v | (u, v) \in A\}$ and the in neighborhood of u is the set $N^-(u) = \{v | (v, u) \in A\}$. Let S be a set of vertices of D. We denote by out neighborhood of S, the set $N^+(S) = (\bigcup_{v \in S} N^+(v)) \setminus S$ and by in neighborhood of S we denote the set $N^-(S) = (\bigcup_{v \in S} N^-(v)) \setminus S$.

A *walk* in the digraph D is a sequence $W = v_1, \ldots, v_t$ of vertices such that $(v_i, v_{i+1}) \in A$ for every $1 \leq i \leq t - 1$ and it is called a walk from v_i to v_t in D. The *length* of this walk is $t - 1$. A walk in which any vertex occurs at most once is called a *path*. A walk where the first vertex is same as the last vertex and all the other vertices are distinct is called a *cycle*. The walks $v_i, v_{i+1}, \ldots, v_j$, $1 \leq i \leq t$, $i \leq j \leq t$ are called subwalks of the walk W. If W is a path these walks are called subpaths of W.

A vertex u is said to be *reachable* from a vertex v in G, if there is a path from u to v in G. The graph G is said to be *strongly connected* if there is a path between every pair of vertices in G.

A graph $D' = (V', A')$ is called a subgraph of D if $V' \subseteq V$ and $A' \subseteq A$. We say D' is a subgraph of D induced by the vertex set $S \subseteq V$ if $V' = S$ and $A' = \{(u, v) \in A | u, v \in S\}$ and denote it by $D[S]$. We say that D' is a subgraph of D induced by the arc set $S \subseteq A$ if $A' = S$ and $V' = \{v \in V |$ there is an arc of M incident on $v\}$ and denote it by $D[S]$.

A set S of vertices (or arcs) with some property is said to be an inclusionwise maximal set with that property if there is no set S' of vertices (respectively arcs) such that $S' \supset S$ and S' has the same property and a set S of vertices (or arcs) with some property is said to be an inclusionwise minimal set with that property if there is no set S' of vertices (respectively arcs) such that $S' \subset S$ and S' has the same property.

Formulas. We assume an infinite supply of propositional *variables*. A *literal* is a variable x or a negated variable $\bar{x}$; if $y = \bar{x}$ is a literal, then we write $\bar{y} = x$. For a set

S of literals we put $\bar{S} = \{\, \bar{x} : x \in S \,\}$; S is *consistent* if $S \cap \bar{S} = \emptyset$. A *clause* is a finite consistent set of literals; we consider a clause as a disjunction of its literals. A finite set of clauses is a *CNF formula* (or *formula*, for short); we consider a formula to be the conjunction of its clauses. A variable x *occurs* in a clause C if $x \in C \cup \bar{C}$; var(C) denotes the set of variables which occur in C. For a set X of variables, lit(X) denotes the set of literals of the variables in X , that is lit$(X) = X \cup \bar{X}$ and for a set L of literals, var(L) denotes the set of variables whose literals are in L, that is var$(L) = \{\, x : x \in L \text{ or } \bar{x} \in L \,\}$. A variable x *occurs* in a formula F if it occurs in one of its clauses, and we let var$(F) = \bigcup_{C \in F}$ var(C) and lit$(F) =$ var$(F) \cup \overline{\text{var}(F)}$. The *length* of a CNF formula F, denoted by $\|F\|$, is defined as $\sum_{C \in F} |C|$.

By $\mathcal{C}(F)$ we denote the set of clauses of a CNF formula F. A formula is *Horn* if each of its clauses contains at most one positive literal, a formula is *Krom* (or *2CNF*, or *quadratic*) if each clause contains at most two literals. The *length* of a CNF formula F is defined as $\sum_{C \in F} |C|$.

If F is a formula and X a set of variables, then we denote by $F \setminus X$ the formula obtained from F after removing all literals in lit(X) from the clauses in F. If $X = \{x\}$ we simply write $F \setminus x$ instead of $F \setminus \{x\}$.

Let F be a formula and $X \subseteq$ var(F). A *truth assignment* is a mapping $\tau : X \to \{\, 0, 1 \,\}$ defined on some set X of variables; we write var$(\tau) = X$. For $x \in$ var(τ) we define $\tau(\bar{x}) = 1 - \tau(x)$. For a truth assignment τ and a formula F, we define $F[\tau] = \{\, C \setminus \tau^{-1}(0) : C \in F,\ C \cap \tau^{-1}(1) = \emptyset \,\}$, i.e., $F[\tau]$ denotes the result of instantiating variables according to τ and applying standard simplifications. A truth assignment τ *satisfies* a clause C if C contains some literal x with $\tau(x) = 1$; τ satisfies a formula F if it satisfies all clauses of F. A formula is *satisfiable* if it is satisfied by some truth assignment; otherwise it is *unsatisfiable*.

3 Important Separators: Intuition and Applications

The notion of important separators was formally introduced in [73] to handle the MULTIWAY CUT problem and the same concept was used implicitly in [16] to give an improved algorithm for the same problem. Subsequently, Chen et al. [17] used this idea to resolve the fixed parameter tractability of the DIRECTED FEEDBACK VERTEX SET problem and Razgon and Barry O' Sullivan [90] proved the fixed parameter tractability of the ALMOST 2-SAT problem using this concept. Some of the more recent applications of this notion have been in FPT algorithms for DIRECTED MULTIWAY CUT[19], MULTICUT [79, 10], ABOVE GUARANTEE VERTEX COVER [88] , SUBSET DIRECTED FEEDBACK VERTEX SET [18], PARITY MULTIWAY CUT [68], and MULTICUT ON DAGS [63]. In this chapter, we describe this simple yet powerful notion and describe a framework using which we will describe the algorithms that come later in the thesis.

Organization of the Chapter In Section 3.1 we give a brief description of the intuition behind the concept of important separators. In Section 3.2, we first present the definition of important vertex separators in undirected graphs, some basic lemmata stemming from these definitions, present an upper bound on the number of important separators of bounded size and give the algorithm of Chen et al. to enumerate all such important separators. We then give analogous versions of these results for arc separators in directed graphs. Towards the end of this section, we show that the bound on the number of important separators proved earlier is essentially tight. Finally, Section 3.3 contains an *important separator template* which allows us to describe the algorithms in the thesis in a modular fashion. We also give an illustration of this template using the MULTIWAY CUT problem as an example.

3.1 Motivation behind important separators

Given a graph $G = (V, E)$ disjoint vertex sets X and Y, suppose we are asked to find a minimum set of vertices in G whose removal disconnects X from Y and also disconnects every pair of vertices in Y. In such cases, if chosen carefully, the set of vertices which we choose to separate X from Y will also help us in separating some pairs of vertices in Y. Intuitively, "the closer it is to Y, the better is the chance of it separating vertices in Y" and hence some separators seem to be more *important* than others. The sets $S_1 = \{s_1, s_2, s_3\}$ and $S_2 = \{s_3, s_4, s_5\}$ (see Fig. 3.1) are sets of the same size which separate X and Y. However, S_2 is *more important* for us since it also separates the vertices in Y, whereas S_1 does not. This intuition was formalized in [73] and used to give an FPT algorithm for the MULTIWAY CUT problem.

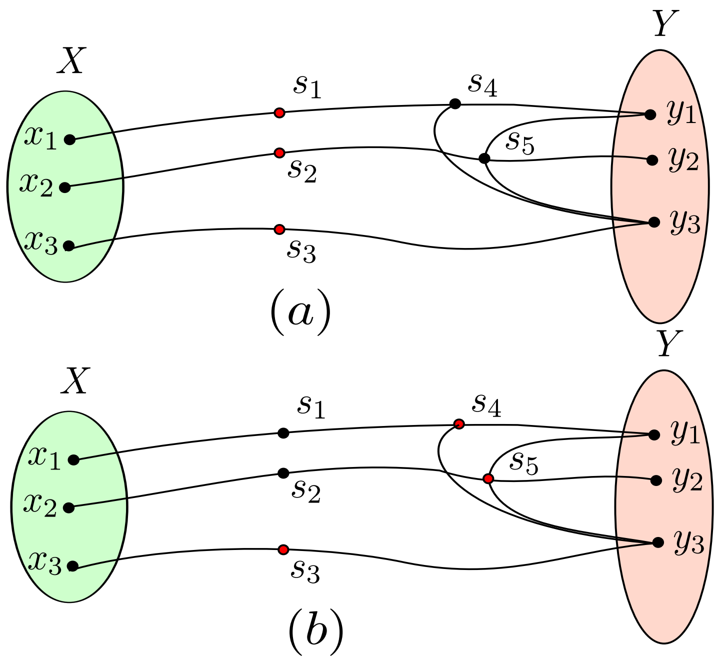

Figure 3.1: $S_1 = \{s_1, s_2, s_3\}$ and $S_2 = \{s_3, s_4, s_5\}$ are two X-Y vertex separators. But S_2 also separates vertices of Y while S_1 does not.

3.2 Important Separators

3.2.1 Separators in Undirected Graphs

Definition 3.2.1. *Let $G = (V, E)$ be an undirected graph and let $X \subseteq V$. We define the function $\tilde{f} : 2^V \to \mathbb{N}$ as $\tilde{f}(X) = |N(X)|$.*

Definition 3.2.2. *Let $G = (V, E)$ be an undirected graph, let $X \subseteq V$ and $S \subseteq V \setminus X$. We denote by $R_G(X, S)$ the set of vertices of G reachable from X in $G \setminus S$ and by $NR_G(X, S)$ the set of vertices of G not reachable from X in $G \setminus S$. We drop the subscript G if it is clear from the context.*

Definition 3.2.3. *Let Z be a finite set. A function $f : 2^Z \to \mathbb{R}$ is submodular if for all subsets A and B of Z, $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$.*

Lemma 3.2.4. *Let $G = (V, E)$ be an undirected graph and let $\tilde{f} : 2^V \to \mathbb{N}$ be a function defined as above. Then the function $\tilde{f}$ is submodular.*

Definition 3.2.5. *Let $G = (V, E)$ be an undirected graph and let $X, Y \subset V$ be two disjoint vertex sets. A subset $S \subseteq V \setminus (X \cup Y)$ is called an X-Y separator in G if $R_G(X, S) \cap Y = \phi$ or in other words there is no path from X to Y in the graph $G \setminus S$. We denote by $\lambda_G(X, Y)$ the size of the smallest X-Y separator in G. An X-Y separator S_1 is said to **cover** an X-Y separator S with respect to X if $R(X, S_1) \supset R(X, S)$. If the set X is clear from the context, we just say that S_1 covers S. An X-Y separator is said to be inclusionwise minimal if none of its proper subsets is an X-Y separator.*

Definition 3.2.6. *Two X-Y separators S and S_1 are said to be **incomparable** if neither covers the other.*

Definition 3.2.7. ([73]) *Let $G = (V, E)$ be an undirected graph and let $X, Y \subset V$ be two disjoint vertex sets. An X-Y separator S_1 is said to **dominate** an X-Y separator S with respect to X if $|S_1| \leq |S|$ and S_1 covers S_2 with respect to X. If the set X is clear from the context, we just say that S_1 dominates S.*

Observation 3.2.8. *Let S_1 and S_2 be two incomparable X-Y separators. Then, $R(X, S_1) \cap S_2 \neq \emptyset$ and $R(X, S_2) \cap S_1 \neq \emptyset$. That is, there is a vertex of S_1 reachable from X in the graph $G \setminus S_2$ and a vertex of S_2 reachable from X in the graph $G \setminus S_1$. Also, $NR(X, S_1) \cap S_2 \neq \emptyset$ and $NR(X, S_2) \cap S_1 \neq \emptyset$. That is, there is a vertex of S_1 separated from X in the graph $G \setminus S_2$ and a vertex of S_2 separated from X in the graph $G \setminus S_1$.*

Lemma 3.2.9. *Let $G = (V, E)$ be a graph and let $s, t \in V$. Let X and Y be two minimal s-t separators such that Y dominates X. Then, Y is also a $(X \setminus Y)$-t separator.*

Proof. This follows from the fact that every vertex in $X \setminus Y$ is in $R(s, Y)$. □

Lemma 3.2.10. *Let $G = (V, E)$ be a graph and let $s, t \in V$. Let X and Y be two minimal s-t separators which are disjoint and incomparable. Let X_{nr} (Y_{nr}) be the set of vertices of X (respectively Y) which are not reachable from s in $G \setminus Y$ (respectively $G \setminus X$), let X_r (Y_r) be the set of vertices of X (respectively Y) which are reachable from s in $G \setminus Y$ (respectively $G \setminus X$). Then, Y_r is a minimal s-X_{nr} separator in the graph $G[R(s, X) \cup X]$.*

Proof. Since we are considering a subgraph of G which does not contain Y_{nr}, Y_r is indeed an s-X_{nr} separator in $G' = G[R(s, X) \cup X]$. We now claim that Y_r is a minimal separator as well. Suppose that this is not the case and let $v \in Y_r$ be such that there is no s-X_{nr} path in G' in $G' \setminus (Y_r \setminus \{v\})$. We know that Y is a minimal s-t separator. Hence, the graph $G \setminus (Y \setminus \{v\})$ contains an s-t path, say P. Since X is an s-t separator, P must intersect X. Consider a traversal of the path P from s to t and let $u \in X$ be the last vertex of X on P. Since $v \in R(s, X)$, u must occur after v in this traversal. We claim that $u \in X_{nr}$. If this were not the case and $u \in X_r$, then we can replace the subpath of P from s to u with another path P_1 from s to u entirely disjoint from Y (P_1 exists by the definition of X_r) to get an s-t path disjoint from Y, which is a contradiction. Hence, we have that $u \in X_{nr}$. In fact, by the same argument, we have that every vertex of X which occurs after v in this traversal must be from X_{nr}. Now, let $x \in X_{nr}$ be the first vertex of X which occurs after v in P. Consider the subpath of P from v to x, call it P_x. Since this path does not intersect X_r, this path also exists in $G' \setminus (Y_r \setminus \{v\})$. Since v lies in $R(s, X)$ and Y is a minimal s-t separator, G' contains a path P_2 from s to v which does not intersect $X \cup (Y_r \setminus \{v\})$. Hence, the concatenation of the paths P_2 and P_x is a walk from s to x in $G' \setminus (Y_r \setminus \{v\})$. Thus, we conclude that Y_r is a minimal s-X_{nr} separator in G'. □

Proposition 3.2.11. *If $R \supseteq X$ is any vertex set disjoint from Y such that $N(R) \cap Y = \phi$*

then $N(R)$ is an X-Y separator.

3.2.2 Important Vertex Separators

Definition 3.2.12. ([73]) *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets and $S \subseteq V \setminus (X \cup Y)$ be an X-Y separator in G. We say that S is an **important** X-Y separator if it is inclusionwise minimal and there does not exist another X-Y separator S_1 such that S_1 dominates S with respect to X. If $S \subset V$ is an important X-Y separator then the set $R(X, S)$ is called an **important set** and the subgraph $G[R(X, S)]$ is called an **important component** if it is connected.*

Lemma 3.2.13. ([73]) *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets. There exists a unique important X-Y separator S^* of size $\lambda_G(X, Y)$.*

Proof. Consider a minimum size X-Y separator of size $\lambda_G(X, Y)$. Since it is minimal, this separator is either important or there is another that dominates it. Hence, there is at least one important X-Y separator of size $\lambda_G(X, Y)$. Now we show that there cannot be two such important X-Y separators.

Suppose S_1 and S_2 are two important X-Y separators of size $\lambda_G(X, Y)$ where $S_1 \neq S_2$ and let $R_1 = R(X, S_1)$ and $R_2 = R(X, S_2)$. We know that $R_1, R_2 \supset X$, and by the minimality of S_1 and S_2, $N(R_1) = S_1$ and $N(R_2) = S_2$. But $S_1, S_2 \cap Y = \phi$. Hence by Proposition 3.2.11 the sets $N(R_1 \cup R_2)$ and $N(R_1 \cap R_2)$ are also X-Y separators and hence $\tilde{f}(R_1 \cup R_2), \tilde{f}(R_1 \cap R_2) \geq \lambda_G(X, Y)$. By the submodularity of $\tilde{f}$ (Lemma 3.2.4), we have that

$$\underbrace{\tilde{f}(R_1)}_{=\lambda_G(X,Y)} + \underbrace{\tilde{f}(R_2)}_{=\lambda_G(X,Y)} \geq \underbrace{\tilde{f}(R_1 \cup R_2)}_{\geq\lambda_G(X,Y)} + \underbrace{\tilde{f}(R_1 \cap R_2)}_{\geq\lambda_G(X,Y)}$$

which implies that $\tilde{f}(R_1 \cup R_2) = \lambda_G(X, Y)$. But this contradicts our assumption that S_1 and S_2 were important X-Y separators since $N(R_1 \cup R_2)$ is an X-Y separator which dominates both S_1 and S_2. $\square$

Note: In the future we will continue to refer to the unique smallest important X-Y separator as S^* without explicit reference to Lemma 3.2.13.

We now show that there is a polynomial time algorithm which computes the unique minimum X-Y separator. We first require the following lemma from [77].

Lemma 3.2.14. (LEMMA 2.4,[77]) *Let s, t be two vertices in a graph $G = (V, E)$ such that the minimum size of an s-t separator is $\ell > 0$. Then, there is a collection $\mathcal{X} = \{X_1, \ldots, X_q\}$ of sets where $\{s\} \subseteq X_i \subseteq V \setminus \{t\}$ such that*

1. *$X_1 \subset X_2 \subset \cdots \subset X_q$,*
2. *$|N(X_i)| = \ell$ for every $1 \leq i \leq q$ and*
3. *every s-t separator of size ℓ is fully contained in $\bigcup_{i=1}^{q} N(X_i)$.*

Furthermore, there is an $\mathcal{O}(\ell(|V|+|E|))$ time algorithm that produces the sets $X_1, X_2 \setminus X_1, \ldots, X_q \setminus X_{q-1}$ corresponding to such a collection $\mathcal{X}$.

Lemma 3.2.15. *Given an undirected graph $G = (V, E)$, disjoint vertex subsets X and Y and a positive integer k, there is an algorithm which in time $\mathcal{O}(k(m + n))$ time either concludes that there is no X-Y separator of size at most k or returns the unique minimum important X-Y separator where $n = |V|$ and $m = |E|$.*

Proof. We can, in $\mathcal{O}(k(m + n))$ time check if the size of the minimum X-Y separator is at most k by running k iterations of the Ford-Fulkerson algorithm [35]. If the size of the minimum separator exceeds k, then we return that there is no X-Y separator of size at most k. Therefore, we assume in the rest of this proof that the size of the minimum X-Y separator is at most k. Let $\mathcal{X} = \{X_1, \ldots, X_q\}$ be a collection with the properties mentioned in Lemma 3.2.14 where "X acts as s and Y acts as t". This can be formalized as follows. We add vertices s and t to the graph G, make $k+1$ copies of each vertex in $X \cup Y$ and add edges from s to all the copies of the vertices in X and from t to all the copies of the vertices in Y. It is clear from this construction that no s-t separator of size at most k will contain any of these newly added vertices and therefore, the s-t separators of size at most k are in one to one correspondence with the X-Y separators of size at most k.

We now claim that $S = N(X_q)$ is the unique minimum important X-Y separator. If this were not the case, then there is an X-Y separator S' which dominates S. Since S is clearly a minimum X_q-Y separator, for every vertex $v \in S$, there is a path from v to Y in the graph $G \setminus X_q$. Furthermore, if S' dominates S, then there is a vertex $v \in (S \cap R(X, S')$. This implies that S' contains a vertex in $V \setminus X_q$, which is a contradiction since Lemma 3.2.14 guarantees that all minimum X-Y separators are contained in the set $\bigcup_{i=1}^{q} N(X_i)$. Therefore, the algorithm simply has to return the set $N(X_q)$. This concludes the proof of the lemma.

$\square$

Lemma 3.2.16. ([79]) *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets and let S be an important X-Y separator. Then $R(X, S) \supseteq R(X, S^*)$.*

Proof. Suppose that this is not the case and let $R_1 = R(X, S)$ and $R_2 = R(X, S^*)$ where $S \neq S^*$. We know that $R_1, R_2 \supset X$ and the minimality of S and S^* implies that $N(R_1) = S$ and $N(R_2) = S^*$. But $S, S^* \cap Y = \phi$. Hence, by Proposition 3.2.11, the sets $N(R_1 \cup R_2)$ and $N(R_1 \cap R_2)$ are also X-Y separators and hence $\tilde{f}(R_1 \cup R_2), \tilde{f}(R_1 \cap R_2) \geq \lambda_G(X, Y)$. By the submodularity of $\tilde{f}$ (Lemma 3.2.4) we have that

$$\tilde{f}(R_1) + \underbrace{\tilde{f}(R_2)}_{=\lambda_G(X,Y)} \geq \tilde{f}(R_1 \cup R_2) + \underbrace{\tilde{f}(R_1 \cap R_2)}_{\geq \lambda_G(X,Y)}$$

which implies that $\tilde{f}(R_1 \cup R_2) \leq \tilde{f}(R_1)$. But this contradicts our assumption that S was an important X-Y separator since $N(R_1 \cup R_2)$ is an X-Y separator which dominates S. $\square$

Lemma 3.2.17. *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets and S be an important X-Y separator.*

(1) S is a $\{v\} - Y$ separator for every $v \in R(X, S)$.

(2) For every $v \in S$, $S \setminus \{v\}$ is an important X-Y separator in $G \setminus \{v\}$.

(3) If S is an $X' - Y$ separator for some $X' \supset X$ such that $G[X']$ is connected, then S is also an important $X' - Y$ separator.

Proof. For the first statement, suppose that this were not the case. Since $v \in R(X, S)$ there is a path from X to v in $G \setminus S$. Now since there is also a path from v to Y, this implies the existence of a path from X to Y in $G \setminus S$. But, this is not possible since S is an X-Y separator.

For the second statement, suppose that $S' = S \setminus \{v\}$ is not an important X-Y separator in $G' = G \setminus \{v\}$. Then there is an X-Y separator S_1 in G' which dominates S' in G'. Consider the set $S_2 = S_1 \cup \{v\}$. Observe that S_2 is also an X-Y separator in G. This is because any path from X to Y which does not contain v exists in G' and hence must contain a vertex of S_2. Now, since S_1 dominates S' in G', S_2 dominates S in G which contradicts our assumption that S is an important X-Y separator.

For the third statement, suppose that this is not the case. Since S is a minimal X-Y separator, S is also a minimal $X' - Y$ separator. Therefore, if S is not an important $X' - Y$ separator it must be the case that there is an $X' - Y$ separator S_1 which dominates

S with respect to X'. We will show that S_1 also dominates S with respect to X, which contradicts our assumption that S is an important X-Y separator. Clearly, $|S_1| \leq |S|$. Hence it is enough for us to show that $R(X, S) \subset R(X, S_1)$.

First we prove that $R(X, S) \subseteq R(X, S_1)$. Consider a vertex v in $R(X, S)$. Clearly $R(X', S) \supseteq R(X, S)$, which means that $v \in R(X', S)$ and since S_1 dominates S with respect to X', v is in $R(X', S_1)$. Since $G[X']$ is connected and contains X, the vertices reachable from X in $G \setminus S_1$ and those reachable from X' in $G \setminus S_1$ are the same implying that $v \in R(X, S_1)$.

Now consider some vertex $u \in S \setminus S_1$. By the minimality of S, u has a neighbor w in $R(X, S)$. But w is also in $R(X, S_1)$ which implies that $u \in R(X, S_1)$ and hence $R(X, S) \subset R(X, S_1)$. $\square$

The following lemma is implicit in [16].

Lemma 3.2.18. ([16]) *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets of G. For every $k \geq 0$ there are at most 4^k important X-Y separators of size at most k. Furthermore, there is an algorithm that runs in time $4^k k(m + n)$ which enumerates all such important X-Y separators, where $n = |V|$ and $m = |E|$.*

Proof. Given $G, X, Y, k \geq 0$ we define a measure $\mu(G, X, Y, k) = 2k - \lambda_G(X, Y)$. We prove by induction on $\mu(G, X, Y, K)$ that there are at most $2^{\mu(G,X,Y,k)}$ important X-Y separators of size at most k.

For the base case, if $2k - \lambda_G(X, Y) < k$ then $\lambda_G(X, Y) > k$ and hence the number of important separators of size at most k is 0. If $\lambda_G(X, Y) = 0$, it means that there is no path from X to Y and hence the empty set alone is the important X-Y separator. For the induction step, consider $G, X, Y, k \geq 0$ such that $\mu = \mu(G, X, Y, k) \geq k$, $\lambda_G(X, Y) > 0$ and assume that the statement of the lemma holds for all G', X', Y', k' where $\mu(G', X', Y', k') < \mu$.

By Lemma 3.2.13, there is a unique important X-Y separator S^* of size $\lambda_G(X, Y)$. Since we have assumed $\lambda_G(X, Y)$ to be positive, S^* is non empty. Consider a vertex $v \in S^*$. Any important X-Y separator S either contains v or does not contain v. For any important X-Y separator S which contains v, $S \setminus \{v\}$ is an important X-Y separator in $G \setminus \{v\}$ (Lemma 3.2.17(2)). Hence the number of important X-Y separators containing v, of size at most k in G is at most the number of important X-Y separators of size at most $k - 1$ in $G \setminus \{v\}$. Observe that $\lambda_{G\setminus\{v\}}(X, Y) = \lambda_G(X, Y) - 1$ which implies that $\mu(G \setminus \{v\}, X, Y, k - 1) < \mu$ and by induction hypothesis, the number of important X-Y separators of size at most $k - 1$ in $G \setminus \{v\}$ is bounded by $2^{\mu-1}$ which is also a bound on the number of important X-Y separators in G which have size at most k and contain v.

Now let S be an important X-Y separator of size at most k which does not contain v. By Lemma 3.2.16 we know that $R(X, S) \supseteq R(X, S^*)$ and by the minimality of S^*, v has a neighbor in $R(X, S)$ which implies that $R(X, S) \supseteq R(X, S^*) \cup \{v\}$. We now set $X' = R(X, S^*) \cup \{v\}$. By Lemma 3.2.17(3) we know that S is a also an important $X' - Y$ separator. Thus a bound on the number of important $X' - Y$ separators of size at most k is also a bound on the number of important X-Y separators of size at most k which do not contain v. First note that $\lambda_G(X', Y) > \lambda_G(X, Y)$ since otherwise we would have an X-Y separator which dominates S^* with respect to X. Now, $\mu(G, X', Y, k) < \mu$ and by induction hypothesis, the number of important $X' - Y$ separators of size at most k is bounded by $2^{\mu-1}$.

Summing up the bounds we get that the number of important X-Y separators of size at most k is bounded by $2 \cdot 2^{\mu-1} = 2^{\mu} \leq 2^{2k}$.

The algorithm follows from the proof above. We apply Lemma 3.2.15 to compute the smallest important X-Y separator S^* or conclude that no such separator of size at most k exists. Then, we pick a vertex v of S^* and recursively enumerate all the important

X-Y separators containing v and those that do not contain v. Since the algorithm of Lemma 3.2.15 takes time $\mathcal{O}(k(m+n))$, the running time follows. This completes the proof of the lemma. $\square$

3.2.3 Arc Separators in Directed Graphs

In this section we consider directed graphs and give analogous definitions and lemmata for arc separators in directed graphs.

Definition 3.2.19. *Let $D = (V, A)$ be a directed graph, let $X \subseteq V$ and $S \subseteq A$. We denote by $R_D(X, S)$ the set of vertices of D reachable from X in $G \setminus S$. We drop the explicit reference to D if it is clear from the context.*

Lemma 3.2.20. *Let $D = (V, A)$ be an undirected graph and let $\hat{f}_a : 2^V \to \mathbb{N}$ be a function defined as $\hat{f}_a(X) = |\delta^+(X)|$. Then the function $\hat{f}_a$ is submodular.*

Definition 3.2.21. *Let $D = (V, A)$ be a directed graph and let $X, Y \subset V$ be two disjoint vertex sets. A subset $S_a \subseteq A$ is called an X-Y arc separator in D if $R(X, S_a) \cap Y = \phi$ or in other words Y is separated from X in $D \setminus S_a$. We denote by $\lambda_D(X, Y)$ the size of the smallest X-Y arc separator in D. An X-Y arc separator S_a^1 is said to **cover** an X-Y arc separator S_a with respect to X if $R(X, S_a^1) \supset R(X, S_a)$. If the set X is clear from the context we just say that S_a^1 covers S_a. An X-Y arc separator is said to be inclusionwise minimal if none of its proper subsets is an X-Y arc separator.*

Definition 3.2.22. *Let $D = (V, A)$ be a directed graph and let $X, Y \subset V$ be two disjoint vertex sets. An X-Y arc separator S_a^1 is said to **dominate** an X-Y arc separator S_a with respect to X if $|S_a^1| \leq |S_a|$ and S_a^1 covers S_a with respect to X. If the set X is clear from the context we just say that S_a^1 dominates S_a.*

Observation 3.2.23. *Observe that any inclusionwise minimal X-Y arc separator does not contain an arc with both endpoints in X.*

Proposition 3.2.24. *If $R \supseteq X$ is any vertex set disjoint from Y then $\delta^+(R)$ is an X-Y arc separator.*

3.2.4 Important Arc Separators

Definition 3.2.25. *Let $D = (V, A)$ be a directed graph, $X, Y \subset V$ be disjoint vertex sets and $S_a \subseteq A$ be an X-Y arc separator in D. We say that S_a is an **important** X-Y arc separator if it is inclusionwise minimal and there does not exist another X-Y arc separator S_a^1 which dominates S_a with respect to X. If $S_a \subset A$ is an important X-Y arc separator then the set $R(X, S_a)$ is called an **important set**.*

Lemma 3.2.26. *Let $D = (V, A)$ be a directed graph, $X, Y \subset V$ be disjoint vertex sets. There exists a unique important X-Y arc separator S_a^* of size $\lambda_D(X, Y)$.*

Lemma 3.2.27. *Let s, t be two vertices in a digraph $D = (V, A)$ such that the minimum size of an s-t separator is $\ell > 0$. Then, there is a collection $\mathcal{X} = \{X_1, \dots, X_q\}$ of sets where $\{s\} \subseteq X_i \subseteq V \setminus \{t\}$ such that*

1. *$X_1 \subset X_2 \subset \cdots \subset X_q$,*
2. *X_i is reachable from s in $D[X_i]$,*
3. *$|\delta^+(X_i)| = \ell$ for every $1 \leq i \leq q$ and*
4. *every s-t separator of size ℓ is fully contained in $\bigcup_{i=1}^{q} \delta^+(X_i)$.*

Furthermore, there is an $\mathcal{O}(\ell(|V| + |A|))$ time algorithm that produces the sets $X_1, X_2 \setminus X_1, \dots, X_q \setminus X_{q-1}$ corresponding to such a collection $\mathcal{X}$.

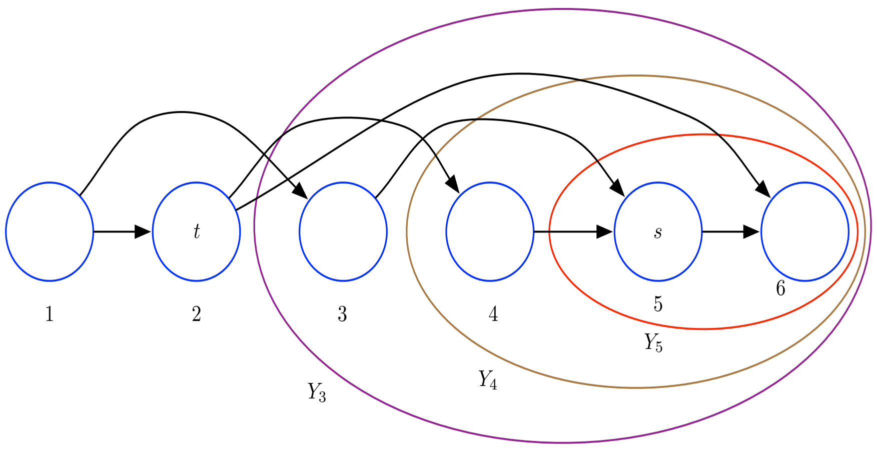

Figure 3.2: An illustration of the sets in the proof of Lemma 3.2.27. The blue circles are the strongly connected components of D_1 and $\alpha(s) = 5$ and $\alpha(t) = 2$.

Proof. This proof is from [77]. However, the proof in [77] does not need that X_i is reachable from s in $D[X_i]$. Thus, we need to do a bit more extra work for the version presented here. We first run ℓ iterations of the Ford-Fulkerson algorithm on the graph with unit capacities on all arcs to find a maximum s-t flow. Let D_1 be the residual graph. Let $C_1, \ldots, C_q$ be a topological ordering of the strongly connected components of D_1 such that $i < j$ if there is a path from C_i to C_j. Recall that there is a t-s path in D_1. Let C_x and C_y be the strongly connected components of D_1 containing t and s respectively. Since there is a path from t to s in D_1, $x < y$. For each $x < i \leq y$, let $Y_i = \bigcup_{j=i}^{q} C_j$ (see Figure 3.2). We first show that $|\delta^+(Y_i)| = \ell$. Since no arcs leave Y_i in the graph D_1, no flow enters Y_i and every arc in $\delta^+(Y_i)$ is saturated by the maximum flow. Therefore, $|\delta^+(Y_i)| = \ell$.

We now show that every arc which is part of a minimum s-t separator is contained in $\bigcup_{i=1}^{q} \delta^+(Y_i)$. Consider a minimum s-t separator S and an arc $(a, b) \in S$. Let Y be the set of vertices reachable from s in $D \setminus S$. Since F is a minimum s-t separator, it must be the case that $\delta^+(Y) = F$ and therefore, $\delta^+(Y)$ is saturated by the maximum flow.

Therefore, we have that (b, a) is an arc in D_1. Since no flow enters the set Y, there is no cycle in D_1 containing the arc (b, a) and therefore, if the strongly connected component containing b is C_{i_b} and that containing a is C_{i_a}, then $i_b < i_a$. Furthermore, since there is flow from s to a from b to t, it must be the case that $x < i_b < i_a < y$ and hence the arc (a, b) appears in the set $\delta^+(Y_{i_a})$.

Finally, we define the set $R(Y_i)$ to be the set of vertices of Y_i which are reachable from s in the graph $D[Y_i]$. Then, the sets $R(Y_y) \subset R(Y_{y-1}) \subset \cdots \subset R(Y_{x+1})$ form a collection of the kind described in the statement of the lemma.

In order to compute these sets, we first need to run the Ford-Fulkerson algorithm for ℓ iterations and perform a topological sort of the strongly connected components of D_1. This takes time $\mathcal{O}(\ell(|V| + |A|))$. During this procedure, we also assign indices to the strongly connected components in the manner described above, that is, $i < j$ if C_i occurs before C_j in the topological ordering. In $\mathcal{O}(\ell(|V| + |A|))$ time, we can assign indices to vertices such that the index of a vertex v (denoted by $\alpha(v)$) is the index of the strongly connected component containing v. We then perform the following preprocessing for every vertex v such that $\alpha(v) < \alpha(s)$. We go through the list of in-neighbors of v and find

$$\beta(v) = \max_{u \in N^-(v)} \Big\{ \alpha(u) \mid \alpha(v) < \alpha(u) \Big\} \text{ and}$$

$$\gamma(v) = \min_{u \in N^-(v)} \Big\{ \alpha(u) \mid \alpha(v) < \alpha(u) \Big\}$$

and set $\beta'(v) = \min\{\beta(v), \alpha(s)\}$ and $\gamma'(v) = \max\{\gamma(v), \alpha(t) + 1\}$.

The meaning of these numbers is that the vertex v occurs in each of the sets $N^+(Y_{\beta'(v)})$, $N^+(Y_{\beta'(v)-1}), \ldots, N^+(Y_{\gamma'(v)})$. This preprocessing can be done in time $\mathcal{O}(m + n)$ since we only compute the maximum and minimum in the adjacency list of each vertex. A vertex v is said to be i-forbidden for all $\gamma'(v) \leq i \leq \beta'(v)$. We now describe the algorithm to compute the sets in the collection.

Computing the collection. We do a modified (directed) breadth first search (BFS) starting from s by using only *out-going* arcs. Along with the standard BFS queue, we also maintain an additional *forbidden* queue.

We begin by setting $i = \alpha(s)$ and start the BFS by considering the out-neighbors of s. We add a vertex to the BFS queue only if it is both unvisited and not i-forbidden. If a vertex is found to be i-forbidden (and it is not already in the forbidden queue), we add this vertex to the forbidden queue. Finally, when the BFS queue is empty and every unvisited out-neighbor of every vertex in this tree is in the forbidden queue, we return the set of vertices *added* to the BFS tree in the current iteration as $R(Y_i) \setminus R(Y_{i+1})$. Following this, the vertices in the forbidden queue which are not $(i-1)$-forbidden are removed and added to the BFS queue and the algorithm continues after decreasing i by 1. The algorithm finally stops when $i = \alpha(t)$.

We claim that this algorithm returns each of the sets $R(Y_{\alpha(s)})$, $R(Y_{\alpha(s)-1}) \setminus R(Y_{\alpha(s)})$, $\ldots, R(Y_{\alpha(t)+1}) \setminus R(Y_{\alpha(t)+2})$ and runs in time $\mathcal{O}(\ell(|V| + |A|)$. In order to bound the running time, first observe that the vertices which are i-forbidden are exactly the vertices in the set $N^+(R(Y_i))$ and therefore the number of i-forbidden vertices for each i is at most ℓ. This implies that the number of vertices in the forbidden queue at any time is at most ℓ. Hence, testing if a vertex is i-forbidden or already in the forbidden queue for a fixed i can be done in time $\mathcal{O}(\ell)$. Therefore, the time taken by the algorithm is $\mathcal{O}(\ell)$ times the time required for a BFS in D, which implies a bound of $\mathcal{O}(\ell(|V| + |A|))$.

For the correctness, we prove the following invariant for each iteration. Whenever a set is returned in an iteration,

- the set of vertices currently in the forbidden queue are exactly the i-forbidden vertices

- the vertices in the current BFS tree are exactly the vertices in the set $R(Y_i)$.

For the first iteration, this is clearly true. We assume that the invariant holds at the end of iteration $j \geq 1$ (where $i = i'$) and consider the $(j+1)$-th iteration (where i is now set as $i' - 1$).

Let P_j be the vertices present in the BFS tree at the end of the j-th iteration and P_{j+1} be the vertices present in the BFS tree at the end of the $(j+1)$-th iteration. We claim that the set $P_{j+1} = R(Y_{i'-1})$.

Since we never add a vertex to P_{j+1} if it is $(i'-1)$-forbidden, the vertices in $P_{j+1} \setminus P_j$ are precisely those vertices which are reachable from P_j via a path disjoint from $(i'-1)$-forbidden vertices. Since the invariant holds for the preceding iteration, we know that $P_j = R(Y_{i'})$ and by our observation about $P_{j+1} \setminus P_j$, we have that P_{j+1} is the set of vertices reachable from $R(Y_{i'})$ via paths disjoint from $(i'-1)$-forbidden vertices, which implies that $P_{j+1} = R(Y_{i'-1})$ since $R(Y_{i'-1})$ is precisely the set of vertices reachable from $R(Y_{i'})$ via paths disjoint from $(i'-1)$-forbidden vertices. We now show that the vertices in the forbidden queue are exactly the $(i'-1)$-forbidden vertices. Since the BFS tree in iteration $j+1$ could not be grown any further, every out-neighbor of every vertex in the tree is in the forbidden queue. Since we have already shown that the vertices in the BFS tree, that is in P_{j+1}, are precisely the vertices in $R(Y_{i'-1})$, we have that every $(i'-1)$-forbidden vertex is already in the forbidden queue. This proves that the invariant holds in this iteration as well and completes the proof of correctness of the algorithm. $\square$

Lemma 3.2.28. *Given an directed graph $D = (V, A)$, disjoint vertex subsets X and Y and a positive integer k, there is an algorithm which in time $\mathcal{O}(k(m+n))$ time either concludes that there is no X-Y arc separator of size at most k or returns the unique minimum important X-Y arc separator where $n = |V|$ and $m = |E|$.*

Proof. We can, in $\mathcal{O}(k(m+n))$ time check if the size of the minimum X-Y arc separator is at most k by running k iterations of the Ford-Fulkerson algorithm [35]. If the size of the minimum arc separator exceeds k, then we return that there is no X-Y arc separator of size at most k. Therefore, we assume in the rest of this proof that the size of the minimum X-Y arc separator is at most k. Let $\mathcal{X} = \{X_1, \ldots, X_q\}$ be a collection with the properties mentioned in Lemma 3.2.27 where "X acts as s and Y acts as t". This can be formalized as follows. We add vertices s and t to the graph G, make $k+1$ copies of each vertex in $X \cup Y$ and add edges from s to all the copies of the vertices in X and from t to all the copies of the vertices in Y. It is clear from this construction that no s-t separator of size at most k will contain any of these newly added vertices and therefore, the s-t separators of size at most k are in one to one correspondence with the X-Y separators of size at most k.

We now claim that $S = N(X_q)$ is the unique minimum important X-Y arc separator. If this were not the case, then there is an X-Y arc separator S' which dominates S. Since S is clearly a minimum X_q-Y arc separator, for every vertex $v \in \textbf{Head}(S)$, there is a path from v to Y in the graph $G \setminus X_q$. Furthermore, if S' dominates S, then there is a vertex $v \in (\textbf{Head}(S) \cap R(X, S'))$. This implies that S' contains an arc in $A[V \setminus X_q] \setminus S$, which is a contradiction since Lemma 3.2.27 guarantees that all minimum X-Y arc separators are contained in the set $\bigcup_{i=1}^{q} \delta^+(X_i)$. Therefore, the algorithm simply has to return the set $\delta^+(X_q)$. This concludes the proof of the lemma. □

Lemma 3.2.29. *Let $D = (V, A)$ be a directed graph, $X, Y \subset V$ be disjoint vertex sets and let S_a be an important X-Y arc separator. Then $R(X, S) \supseteq R(X, S_a^*)$.*

Lemma 3.2.30. *Let $D = (V, A)$ be a directed graph, $X, Y \subset V$ be disjoint vertex sets and S_a be an important X-Y arc separator.*

1. *For every $v \in R(X, S_a)$, S_a is a $\{v\} - Y$ arc separator.*

2. *For every arc $e \in S_a$, $S_a \setminus \{e\}$ is an important X-Y arc separator in $D \setminus \{e\}$.*

3. *If S_a is an $X' - Y$ arc separator for some $X' \supset X$ such that X' is reachable from X in the induced subgraph $D[X']$, then S_a is an important $X' - Y$ arc separator.*

Proof. For the first statement, suppose this were not the case. Since $v \in R(X, S)$ there is a path from X to v in $G \setminus S$. Now since there is also a path from v to Y, this results in a walk from X to Y which can be converted to a path from X to Y in $G \setminus S$. But this is not possible since S is an X-Y arc separator.

For the second statement, suppose that $S'_a = S_a \setminus \{e\}$ is not an important X-Y arc separator in $D' = D \setminus \{e\}$. Then, in D', there is an X-Y arc separator S^1_a which dominates S'_a. Consider the set $S^2_a = S^1_a \cup \{e\}$. Observe that S^2_a is also an X-Y arc separator in D. This is because any path from X to Y which does not contain the arc e, exists in D' and hence must contain an arc in S^2_a. Now, since S^1_a dominates S'_a in D', S^2_a dominates S_a in G which contradicts our assumption that S_a is an important X-Y arc separator.

For the third statement, assume that this is not the case. Clearly S_a is a minimal $X' - Y$ arc separator and hence there is an $X' - Y$ arc separator S^1_a which dominates S_a with respect to X'. We will prove that S^1_a also dominates S_a with respect to X. Since we already have that $|S^1_a| \leq |S_a|$, it is enough for us to show that $R(X, S_a) \subset R(X, S^1_a)$.

First we prove that $R(X, S_a) \subseteq R(X, S^1_a)$. Consider a vertex v in $R(X, S_a)$. Clearly $R(X', S_a) \supseteq R(X, S_a)$. But, $R(X', S_a) \subset R(X', S^1_a)$, which means that $v \in R(X', S^1_a)$. Since X' contains X and is reachable from X in $D[X']$, and S^1_a does not contain an arc with both end points inside X' (by Observation 3.2.23), X' is reachable from X in $D[X'] \setminus S^1_a$ also, and hence $v \in R(X, S^1_a)$.

Now, consider some arc $e = (u, w) \in S_a \setminus S^1_a$. By the minimality of S_a, $u \in R(X, S_a)$ and $w \notin R(X, S_a)$. But we have shown that u is also in $R(X, S^1_a)$, which implies that $w \in R(X, S_a^1)$ and hence $R(X, S_a) \subset R(X, S_a^1)$.

□

Lemma 3.2.31. *Let $D = (V, A)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets of D. For every $k \geq 0$ there are at most 4^k important X-Y arc separators of size at most k. Furthermore,*

Proof. Given $D, X, Y, k \geq 0$ we define a measure $\mu_a(D, X, Y, k) = 2k - \lambda_D(X, Y)$. We prove by induction on $\mu_a(D, X, Y, K)$ that there are at most $2^{\mu_a(D,X,Y,k)}$ important X-Y arc separators of size at most k. For the base case, if $2k - \lambda_D(X, Y) < k$, then $\lambda_D(X, Y) > k$ and hence the number of important X-Y arc separators of size at most k is 0. If $\lambda_D(X, Y) = 0$, it means that there is no path from X to Y and hence the empty set alone is the important X-Y arc separator. Consider $D, X, Y, k \geq 0$ such that $\mu_a = \mu_a(D, X, Y, k) \geq k, \lambda_D(X, Y) > 0$ and assume that the statement holds for all D', X', Y', k' where $\mu_a(D', X', Y', k') < \mu_a$.

By Lemma 3.2.26 there is a unique important X-Y arc separator S_a^* of size $\lambda_D(X, Y)$. Since we have assumed $\lambda_D(X, Y)$ to be positive, S_a^* is non empty. Consider an arc $e = (u, v) \in S^*$. Any important X-Y arc separator S either contains e or does not contain e. For any important X-Y arc separator S_a which contains e, $S_a \setminus \{e\}$ is an important X-Y arc separator in $D \setminus \{e\}$ by Lemma 3.2.30(2). Hence the number of important X-Y arc separators of size at most k in D which contain e, is at most the number of important X-Y arc separators of size at most $k - 1$ in $D \setminus \{e\}$. Observe $\lambda_{D\setminus\{e\}}(X, Y) = \lambda_D(X, Y) - 1$ which implies that $\mu_a(D \setminus \{e\}, X, Y, k - 1) < \mu_a$ and by induction hypothesis, the number of important X-Y arc separators of size at most $k - 1$ in $D \setminus \{e\}$ is bounded by $2^{\mu_a - 1}$ which is also a bound on the number of important X-Y arc separators of size at most k in D which contain e.

Now let S_a be an important X-Y arc separator of size at most k which does not contain e. By Lemma 3.2.29 we know that $R(X, S_a) \supseteq R(X, S_a^*)$ and by the minimality of S_a^*, $u \in R(X, S_a)$ and $v \notin R(X, S_a)$ which implies that $R(X, S_a) \supseteq R(X, S_a^*) \cup \{v\}$. We now set $X' = R(X, S_a^*) \cup \{v\}$. By Proposition 3.2.30(3) we know that S_a is a also an important $X' - Y$ arc separator. Thus a bound on the number of important $X' - Y$ arc separators of size at most k is also a bound on the number of important X-Y arc separators of size at most k which do not contain v. First note that $\lambda_D(X', Y) > \lambda_D(X, Y)$ since otherwise we would have an X-Y arc separator which dominates S_a^*. Now, $\mu_a(D, X', Y, k) < \mu_a$ and by induction hypothesis, the number of important $X' - Y$ arc separators of size at most k is bounded by $2^{\mu_a - 1}$.

Summing up the bounds we get that the number of important X-Y arc separators of size at most k is bounded by $2 \cdot 2^{\mu_a - 1} = 2^{\mu_a} \leq 2^{2k}$.

The algorithm follows from the proof of above. We apply Lemma 3.2.28 to compute the smallest important X-Y separator S^* or conclude that no such separator of size at most k exists. Then, we pick a vertex v of S^* and recursively enumerate all the important X-Y separators containing v and those that do not contain v. Since the algorithm of Lemma 3.2.28 takes time $\mathcal{O}(k(m + n))$, the running time follows. This completes the proof of the lemma. $\square$

3.2.5 Tight Instances for the Bound on Number of Important Separators

In this section we demonstrate instances which have a *large* number of important separators, thus proving that the bounds stated in the previous sections are essentially tight.

Lemma 3.2.32. *The bound of 4^k on the number of important X-Y separators and important X-Y arc separators is tight up to a polynomial factor of k.*

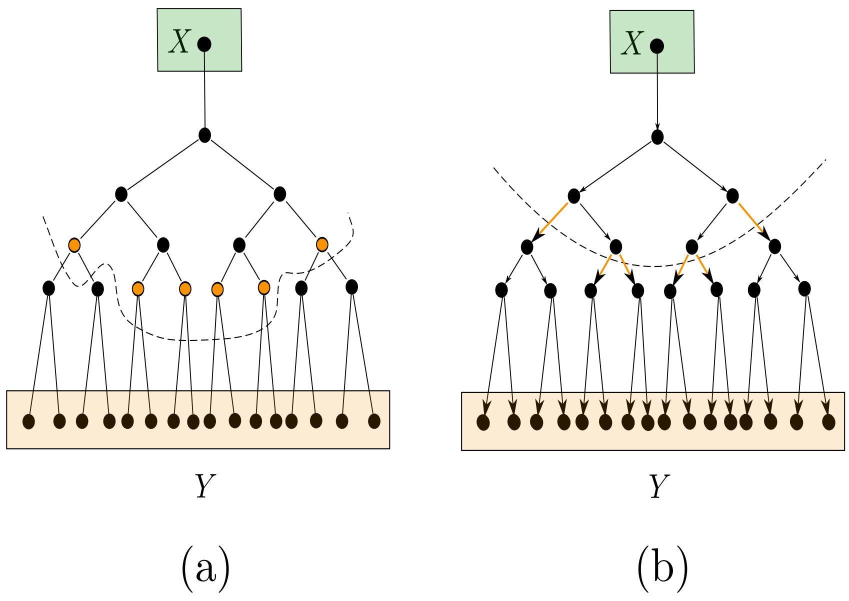

Figure 3.3: Graphs which achieve the given bound on important (arc) separators upto a polynomial factor of k.

Proof. In the proof of Lemma 3.2.18 we have shown that the number of important X-Y separators is bounded by the sum of the important separators which contain and those which do not contain some vertex of the smallest important X-Y separator. This is sum is clearly maximized when the measure μ decreases precisely by 1 in either branch. Based on this idea, we demonstrate instances (see Fig. 3.3(1)) where the number of important X-Y separators is at least $4^k/poly(k)$ where $poly(k)$ is some polynomial function of k.

For the instance shown in Fig. 3.3(1), any minimal X-Y separator is an important X-Y separator. Hence the number of important X-Y separators of size at most k is the number of minimal X-Y separators of size at most k. But any minimal X-Y separator of size say p, corresponds to a subtree of T rooted at X and with p leaves. Hence the number of minimal X-Y separators of size at most k is the number of subtrees of T

which are rooted at X and have at most k leaves. But this number is the Catalan number C_{k-1} which is asymptotically $\frac{4^k}{k^{3/2}}$. An analogous arguments holds for important X-Y arc separators. □

Remark 3.2.33. *The example of the tight instances given above can be found in the slides of Dániel Marx titled* Important separators and spiders.

The Important Separator Template

1. Show that either there is a solution for the given instance which contains a minimal X-Y separator of bounded size for two vertex sets X and Y or there is a possible branching/reduction which enables us to guarantee such a solution, or the problem is tractable (FPT).
2. Define a **dominating set** for the set of all minimal X-Y separators of bounded size and show that there is a solution which intersects an X-Y separator in this **dominating set**. Following this, prove the existence of a bounded set of vertices which either intersects the **dominating set** or contains a *branchable* vertex.

   For most problems the **dominating set** is simply the set of all important X-Y separators of a particular size. However, in this thesis, we introduce a more intricate notion of **dominating set** depending on the problem at hand.
3. Show that the bounded set of vertices referred to above can be computed in FPT time.
4. Combine the above steps to obtain a branching algorithm.

3.3 The Important Separators Template

In this section, we describe a template for applying the important separator notion to design FPT algorithms. The purpose of presenting this template is two-fold. Firstly, it serves to isolate and describe the novel features of some of the algorithms which we introduce in the thesis. Secondly, it provides a useful alternate perspective on the existing FPT algorithms for graph separation problems since they all follow this template to a certain extent.

3.3.1 Applying the template to Multiway Cut

We explain the algorithm for MULTIWAY CUT given by Marx and Chen et al. using the above template. However, for ease of description, we describe an algorithm with a worse dependence on the parameter compared to that given by Chen et al.

Let $(G = (V, E), T, k)$ be the given instance of MULTIWAY CUT and let $t_1 \in T$ where $|V| = n$ and $|E| = m$.

1. Any optimum solution contains a minimal t_1-$T \setminus t_1$ separator of size at most k. Otherwise we are already done.

2. The set of important t_1-$T \setminus t_1$ separators of size at most k forms a **dominating set** [73], that is, there is a solution for the given instance which contains an important t_1-$T \setminus t_1$ separator of size at most k . The number of such important separators is bounded by 4^k (Lemma 3.2.18) and therefore the union of the vertices in the important separators is a set which intersects some solution for the given instance and has size at most $4^k k$.

3. The set of important t_1-$T \setminus t_1$ separators of size at most k can be computed in time $\mathcal{O}(4^k k(m + n))$ (Lemma 3.2.18).

4. Branch on the vertices in the union of the important t_1-$T \setminus t_1$ separators of size at most k.

The above algorithm results in a search tree of depth at most k where each internal node has at most $4^k k$ children, implying a search tree with at most $2^{2k^2+k\log k}$ leaves. Since the time spent at each internal node is bounded by $\mathcal{O}(4^k k(m+n))$, this algorithm runs in time $\mathcal{O}(2^{2k^2+k+\log k}k(m+n))$.

Part II

Above Guarantee Parameterizations of Vertex Cover

4

Matchings and Vertex Covers

4.1 The Maximum Matching Problem

The classical notions of *matchings* and *vertex covers* have been at the center of serious study for several decades in the area of Combinatorial Optimization [70]. In 1931, König and Egerváry independently proved the following result of fundamental importance

Theorem 4.1.1. ([70]) *In a bipartite graph, the size of the maximum matching is equal to the size of the minimum vertex cover.*

This led to a polynomial-time algorithm for finding a minimum vertex cover in bipartite graphs. Interestingly, this min-max relationship holds for a larger class of graphs known as König-Egerváry graphs (or König graphs) and it includes bipartite graphs as a proper subclass (see Fig. 4.1). In this section, we recall some classical theorems on the subject of matchings and the definitions of the combinatorial objects used in the majority of these results.

Definition 4.1.2. *Given a graph $G = (V, E)$ and a matching M, we call a path $P =$*

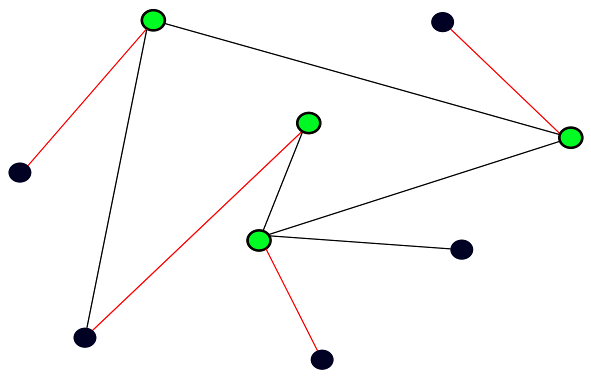

Figure 4.1: A non-bipartite König graph where the size of the minimum vertex cover (green vertices) is equal to the size of the maximum matching (red edges).

*$v_1, \ldots, v_t$ in the graph, an M-**alternating path** (or **alternating path** if M is clear from the context) if for every $2 \leq i \leq t-1$ in P, either $(v_{i-1}, v_i) \in M$ or $(v_i, v_{i+1}) \in M$ but not both.*

The following theorem of Berge is a fundamental result relating maximum matchings and alternating paths.

Theorem 4.1.3. ([4]) *A matching M in a graph G is not of maximum cardinality if and only if there is an alternating path between two vertices left unsaturated by M*

Theorem 4.1.3 led to a polynomial time algorithm to compute a maximum matching in a bipartite graph.

Theorem 4.1.4. *Maximum matching in a bipartite graph can be computed in polynomial time.*

Definition 4.1.5. *Given a graph $G = (V, E)$ and a matching M, we call an alternating path $P = v_1, \ldots, v_t$ in the graph, an **odd** M-path from v_1 to v_t (or from v_t to v_1) if the edges (v_1, v_2) and (v_{t-1}, v_t) are in M (see Fig 4.2) and an **even** M-path from v_1 to v_t if*

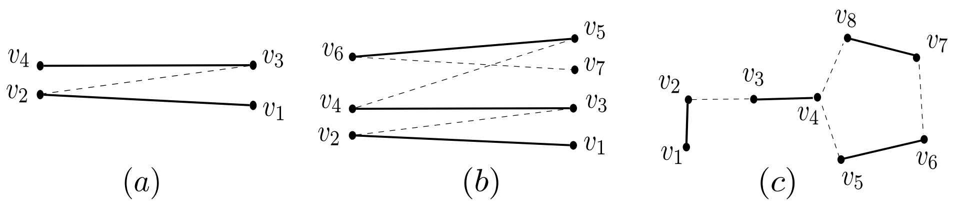

Figure 4.2: Illustrations of the two types of M-alternating paths and an M-flower. The non matched edges are represented by the dashed lines. (a) An odd M-path v_1, v_2, v_3, v_4. (b) An even M-path $v_1, v_2, v_3, v_4, v_5, v_6, v_7$. (c) An M-flower with root v_1, base v_4, stem v_1, v_2, v_3, v_4, blossom $v_4, v_5, v_6, v_7, v_8, v_4$, and a blossom path v_5, v_6, v_7, v_8.

the edge (v_1, v_2) is in M, but the edge (v_{t-1}, v_t) is not. If $v_1 \in X$ and $v_t \in Y$, then P is also called an odd (respectively even) M-path from X to Y.

Note that, by our definition, a single matched edge is an odd M-path. In addition, we consider a path consisting of a single vertex to be an even M-path.

Definition 4.1.6. *An odd cycle $C = v_1, v_2, \ldots, v_t, v_1$ is called an M-**blossom** (or **blossom** if M is clear from the context) if the path $v_2, \ldots, v_t$ is an odd M-path. The vertex v_1 is called the **base** of the blossom. The odd M-path $v_2, \ldots, v_t$ is called a **blossom M-path** (or **blossom path** if M is clear from the context).*

Definition 4.1.7. *The union of a blossom C and an odd M-path P is called an M-**flower** (or **flower** if M is clear from the context) if the base of C is an endpoint of P and C and P do not intersect in any other vertex. The base of C is also called the **base** of the flower, P is called the **stem** and the endpoint of P disjoint from C is called the **root** of the flower. If the root of the blossom is in a set X, then the flower is referred to as an X-flower.*

Definition 4.1.8. *Given a graph $G = (V, E)$ and a matching M, an M-**alternating tree** (or **alternating tree** if M is clear from the context) rooted at a saturated vertex v*

is defined as the output of the following procedure. Label v even*, and keep all the other vertices unlabeled at this point.*

1. *For each even vertex u, label it's matching partner u'* odd *and make u' a child of u.*

2. *For each odd vertex u, make its unlabeled neighbors the children of u and label them* even.

3. *If neither of the above operations can label an unlabeled vertex, then return the tree constructed.*

The following observations are a consequence of the definition of alternating trees.

Observation 4.1.9. *Given a graph $G = (V, E)$ and a matching M in G, consider the M-alternating tree rooted at a saturated vertex $v \in V$.*

(a) *There is an odd (even) M-path from v to a vertex u if and only if u is labeled* odd *(respectively* even*) in this tree.*

(b) *G has an M-flower rooted in v if and only if there is an edge between two vertices labeled odd in this tree.*

Observation 4.1.10. *Given a graph $G = (V, E)$, a matching M, and a vertex u, we can test in time $\mathcal{O}(m)$ if there is an M-flower in G with u as the root. Furthermore, given a vertex v, we can test in time $\mathcal{O}(m)$ if there is an odd (or even) M-path from u to v in G.*

Proof. This follows from the fact that the alternating tree rooted at u can be computed in time $\mathcal{O}(m)$ assuming that the matching has been given in the form of an appropriate data structure. Since this is straightforward, we do not give a detailed proof. $\square$

4.2 König Graphs with Extendable Vertex Covers

In this section we obtain an extension of the result by König and Egerváry relating maximum matchings and minimum vertex covers. More precisely, we address the following question.

> *When does a König graph have a minimum vertex cover containing a specified subset of vertices of the graph?*

While the *question* can be considered an extension of that studied by König and Egerváry, the *answer* we obtain revolves around the structures of alternating paths and flowers (see previous section) used by Berge [4] and Edmonds [29] in their classical results.

We begin by giving a sketch of our characterization. Consider the path $P_\ell = v_1, \ldots, v_\ell$, where ℓ is even. It is clearly a König graph since the size of the maximum matching is $\frac{\ell}{2}$ and we have a vertex cover of size $\frac{\ell}{2}$ comprising the vertices $v_2, v_4, \ldots, v_\ell$. However, observe that there is no minimum vertex cover which contains v_1 and v_ℓ since any minimum vertex cover containing v_1 is forced to exclude the set $\{v_2, v_4, \ldots, v_\ell\}$. In other words, if there is an odd M-path between two vertices of the König graph, then we cannot have a minimum vertex cover containing both end points of the path. Similarly, consider a graph which is an M-flower. By an argument similar to the previous one (for an odd length path), observe that there cannot be a minimum vertex cover for this graph containing the root.

The above arguments give us an idea as to what type of structures must necessarily be excluded for the given *annotated* König graph to have a minimum vertex cover containing the annotated vertices Our main contribution in this regard is a proof that it is also *sufficient* for the above structures to be absent from the given annotated graph for it to have a minimum vertex cover containing the annotated vertices. Before we present a formal proof of our characterization., we revisit a different characterization of König graphs, followed by some structural observations regarding alternating paths and flowers in König graphs.

Lemma 4.2.1 (see, for example, [81]). *A graph $G = (V, E)$ is König if and only if there exists a bipartition of V as $V_1 \uplus V_2$, with V_1 a vertex cover of G such that there exists a matching across the cut (V_1, V_2) saturating every vertex of V_1.*

Due to Lemma 4.2.1, we define a König graph G as a triple (A, B, E) where the vertex set of the graph is $A \cup B$ with A being a minimum vertex cover of G, B the corresponding independent set and E is the edge set of the graph. We also have the following simple observation about König graphs.

Observation 4.2.2. *Given a König graph $G = (A, B, E)$, let M be a maximum matching of G and let U be the set of vertices left unsaturated by M. Then, M saturates A across the cut (A, B) and no vertex of U is present in any minimum vertex cover of G.*

Proof. Since every minimum vertex cover of a König graph contains exactly one vertex of each edge in any maximum matching, it must be the case that M saturates A across the cut (A, B). Furthermore, if a vertex in U occurs in some minimum vertex cover of G, say S, then since any vertex cover is forced to contain at least one vertex from each edge of M, S has size at least $|M| + 1$, which is a contradiction. This completes the proof. $\square$

The following consequences follow from the definitions in the previous section.

Lemma 4.2.3. *Let $G = (A, B, E)$ be a König graph with a maximum matching M and let $L \subseteq A$ and $R \subseteq B$ be vertex subsets.*

(a) There is no odd M-path from A to A.

(b) Any odd M-path from B to B contains exactly one edge between two vertices in A.

(c) There is no blossom with the base in B.

(d) In any blossom in G, one of the two neighbors of the base in the blossom is in B.

(e) A minimum vertex cover of G contains at most 1 endpoint of an odd M-path. Furthermore, if a minimum vertex cover of G does not contain a vertex u, then it does not contain any vertex which has an even M-path to u.

(f) Let $P = v_1, \ldots, v_t$ be an odd M-path from B to B. Then there is an edge (u, v) such that $u, v \in A$ and there is an odd M-path P_1 from v_1 to u and an odd M-path P_2 from v_t to v such that every matched edge in P is present in either P_1 or P_2.

(g) Consider a vertex v and an even M-path P from some vertex u to v. Then P does not contain the matching partner of v.

Proof. (a) Consider the bipartite graph obtained from G by deleting edges with both endpoints in A. If there were an odd M-path from A to A in G, then this path also exists in the bipartite graph which we constructed, implying an odd path between the same partition of the bipartite graph, a contradiction.

(b) We know that any odd M-path from B to B which does not contain an edge between two vertices in A is also an odd path from B to B in the bipartite graph constructed as described in (a), a contradiction. Hence any such path must contain at least one edge between two vertices of A. We now show that there is exactly one such edge. Let $P = v_1, \ldots, v_t$ be the odd M-path under consideration and suppose that there are two edges $e_1 = (v_i, v_{i+1})$ and $e_2 = (v_j, v_{j+1})$ in P such that $v_i, v_{i+1}, v_j, v_{j+1} \in A$. Assume without loss of generality that $i + 1 < j$. Then, the edges $(v_{i+1}, v_{i+2}), (v_{j-1}, v_j)$ must be in M. They cannot be the same edge since that would mean a matched edge between vertices of A. However, the subpath $v_{i+1}, v_{i+2}, \ldots, v_j$ is an odd M-path from A to A , which contradicts (a).

(c) If this were not the case, then there is a blossom with some vertex $b \in B$ as its base. Let u_1 and u_2 be the neighbors of b in the blossom. Since B is independent, $u_1, u_2 \in A$. However, by definition, the blossom path between u_1 and u_2 is an odd M-path, a contradiction to (a).

(d) Let u_1 and u_2 be the neighbors of the base of the blossom within the blossom. Suppose $u_1, u_2 \in A$. Then the blossom path from u_1 to u_2 is an odd M-path from A to A which contradicts (a).

(e) Let $P = v_1, \ldots, v_t$ be an odd M-path and let S be a minimum vertex cover for G. Note that since S is a minimum vertex cover for G, S must contain exactly one end point of each matched edge in M. We prove by induction on t that if $v_1 \in S$, then $v_t \notin S$. In the base case, when $t = 2$, the claim is clearly true. Therefore, let $t > 2$ and assume that our claim is true $\forall t' < t$. Consider the path $v_1, \ldots, v_{t-2}$. It is clearly an odd M-path. By the induction hypothesis, $v_1 \in S$ implies that $v_{t-2} \notin S$. Since S is a vertex cover, it must be the case that $v_{t-1} \in S$ in order to cover the edge (v_{t-2}, v_{t-1}). However, for S to be a minimum vertex cover, it must contain exactly one vertex from the matched edge (v_{t-1}, v_t), implying that $v_t \notin S$. For the proof of the second statement, suppose that G contains a minimum vertex cover S excluding u and let v a vertex such that there is an even M-path P from v to u. The statement is clearly true if $u = v$. Therefore, let $P = v_1, \ldots, v_t$ be this path where $v = v_1$ and $u = v_t$ and $t > 1$. Since v_t is not in S, it must be the case that $v_{t-1} \in S$. Since P is an even M-path, the subpath of P from v_1 to v_{t-1} is an odd M-path and by the first statement, $v_{t-1} \in S$ implies that $v_1 \notin S$. Therefore, we have that v is also disjoint from S.

(f) Let $P = v_1, \ldots, v_t$ be an odd M-path from B to B. Let v_i and v_{i+1} be the vertices occurring in P such that $v_i, v_{i+1} \in A$ given by (b). Let P_1 be the subpath of P from v_1 to v_i and let P_2 be the subpath of P from v_{i+1} to v_t. We claim that P_1 and P_2 are both odd M-paths. Note that $v_i, v_{i+1} \in A$ implies that $(v_i, v_{i+1}) \notin M$. Since P is an M-alternating path, the edges (v_{i-1}, v_i) and (v_{i+1}, v_{i+2}) are in M. Therefore, P_1 is an M-alternating path where the first and last edges are matched edges, that is, P_1 is an odd M-path. The same is the case for P_2 and since P_1 and P_2 together contain every matched edge in P, the claim follows.

(g) Let v' be the matching partner of v and suppose P contains v'. Observe that $u \neq v'$ since that would imply that the first edge of P is a matched edge, implying that the first edge of P is (v', v), which is a contradiction to our assumption that P is an even M-path from u to v. Since v' is not an end point of P, v' has two edges of P incident on it. Since P is an alternating path, at least one of these two edges is that edge (v', v). Since v is an endpoint of P, it must be the case that the last edge of P is the edge (v', v). Since (v', v) is a matched edge, it contradicts our assumption that P is an even M-path from u to v.

$\square$

Using the above observations, we prove our main structural result.

Lemma 4.2.4. *Let $G = (A, B, E)$ be a König graph with a maximum matching M, let $L \subseteq A$ and $R \subseteq B$ be saturated vertex subsets, and let $U \subseteq B$ be the set of vertices of G left unsaturated by M.Then, G has a minimum vertex cover containing $L \cup R$ if and only if G contains none of the following structures.*

1. *an odd M-path from $L \cup R$ to $L \cup R$*

2. *an even M-path from $L \cup R$ to U.*

3. *an R flower*

Proof. ($\Rightarrow$)Let $A = a_1, \ldots, a_r$ and $M = \{m_1, \ldots, m_r\}$ where $m_i = (a_i, b_i)$ and let $U = \{b_{r+1}, \ldots, b_{|B|}\}$. Suppose G has a minimum vertex cover S containing L and R. Consider an odd M-path in G. By Lemma 4.2.3(e), it cannot be the case that both endpoints of this path are in S. Hence, any odd M-path in G has at least one endpoint disjoint from $L \cup R$.

Suppose that G has an even M-path P from $L \cup R$ to a vertex $u \in U$. Since no vertex in U is part of any minimum vertex cover (by Observation 4.2.2), U is disjoint from S. Therefore, by Lemma 4.2.3(e), any vertex which has an even M-path to U must also be disjoint from S. Since $L \cup R \subseteq S$, it cannot be the case that a vertex in $L \cup R$ has an even M-path to a vertex in U.

Finally, consider the case when G has an R-flower. Then, the root of the flower is in S since $R \subseteq S$. By the definition of flowers, the stem is an odd M-path from the root to the base and by Lemma 4.2.3(e) the base is not in S. This implies that S contains both neighbors of the base in the blossom, a contradiction to Lemma 4.2.3(e) since the blossom path between these two vertices is an odd M-path.

($\Leftarrow$) Suppose G has no odd M-paths from $L \cup R$ to $L \cup R$, no even M-paths from $L \cup R$ to U and no R-flowers. We define S_1 as the set of all vertices to which there is an even M-path from R. We define the set S_2 as the set of all vertices a_i such that m_i is not covered by S_1. We let S denote the union of S_1 and S_2. Formally,

$$S_1 = \{v | \text{ there is an even } M\text{-path from } R \text{ to } v\}.$$

$$S_2 = \{a_i | V(m_i) \cap S_1 = \phi\}.$$

$$S = S_1 \cup S_2.$$

We claim that S is a minimum vertex cover of G, containing $L \cup R$. We first prove that S contains $L \cup R$. Recall that by definition, a single vertex is an even M-path and hence S_1 contains R. Suppose there is a vertex a_i in L which is not in S. By the definition of S, $b_i \in S_1$, implying that there is an even M-path P from R to b_i. By Lemma 4.2.3(g) P does not contain a_i. Hence, $P + (b_i, a_i)$ is an odd M-path from R to L, a contradiction to our assumption that there are no odd M-paths from $L \cup R$ to $L \cup R$. Hence, we conclude that $L \cup R \subseteq S$.

We now prove that S is indeed a vertex cover. Suppose that S is not a vertex cover and let $e = (a_i, b_j)$ be an edge left uncovered by S. Note that the definition of S implies that e cannot be a matched edge since if S_1 picked neither endpoint of a matched edge, then we will have added the vertex of that edge lying in A, into S_2 and hence in S as well. Therefore, $i \neq j$. Since $a_i \notin S$, it must be the case that $b_i \in S_1$. Hence there is an even M-path $P = v_1, \ldots, v_t$ from R to b_i where $v_t = b_i$. By Lemma 4.2.3(g) P does not contain a_i.

If P did not contain b_j, then $P + (b_i, a_i) + (a_i, b_j)$ is an even M-path from R to b_j. But this implies that $b_j \in S_1 \subseteq S$, a contradiction to our assumption that e was not covered by S. Therefore, we assume that P contains b_j. Furthermore, since $b_j \notin S$, we have that $b_j \notin R$, which implies that b_j is not an endpoint of P and hence there are two edges incident on b_j in P. Since one of the edges incident on b_j in P is a matched edge, it must be the case that $b_j \notin U$ and b_j has a matching partner a_j. Furthermore, a_j is adjacent to b_j in P. If a_j occurs immediately after b_j, then the subpath of P from

v_1 to b_j is an even M-path, in which case b_j would have been added to S_1, covering the edge e. Hence, a_j must occur immediately before b_j in P. Let P' be the subpath of P from v_1 to b_j. Now, $P' + (b_j, a_i)$ is an even M-path from R to a_i, which implies that $a_i \in S_1 \subseteq S$, a contradiction. We have thus established that S is indeed a vertex cover of G.

We now prove that S is a minimum vertex cover of G. In particular we will prove that S contains exactly one vertex from every edge of M. Since we have already shown that S is a vertex cover, it contains at least one vertex from every matched edge. Furthermore, due to the absence of even M-paths from $L \cup R$ to U, no vertex of U is in S. Therefore it is enough for us to prove that S does not contain both end points of any matched edge. Suppose there is a matched edge m_j such that both a_j and b_j are in S. Then it must be the case that both a_j and b_j are in S_1 as well. This could only be possible if there were even M-paths $P_1 = x_1, \ldots, x_s$ and $P_2 = y_1, \ldots, y_t$ from R to a_j and b_j respectively. For each matched edge m_i such that S contains both end points, let P_1^i and P_2^i be even M-paths of least length from R to a_i and b_i respectively. Among all such matched edges, let m_d be one which minimizes the sum $|P_1^i| + |P_2^i|$, that is $|P_1^d| + |P_2^d| = min_i\{|P_1^i| + |P_2^i|\}$. For the sake of convenience, in the rest of the proof of this lemma, we refer to the paths P_1^d and P_2^d as P_1 and P_2. By Lemma 4.2.3(g), P_1 does not contain b_d and P_2 does not contain a_d. We now consider the following two cases.

1. If P_1 and P_2 do not intersect at all, then $P_1 + (a_j, b_j) + Rev(P_2)$ is an odd M-path from R to R, a contradiction.

2. Suppose P_1 and P_2 do intersect and let $y_q = x_p$ be the last vertex along P_2 when traversing from y_1, which is also present in P_1. Since one of the two edges incident on y_q in P_2 is a matched edge, it must be the case that the edge (y_{q-1}, y_q) is a matched edge. For the same reason, it must be the case that one of the two edges (x_p, x_{p+1})

or (x_{p-1}, x_p) is the same as the edge (y_{q-1}, y_q), that is, either $y_{q-1} = x_{p+1}$ or $y_{q-1} = x_{p-1}$. Since P_1 and P_2 are both disjoint from either a_d or b_d, the edge (y_{q-1}, y_q) cannot be the edge m_d since it occurs in both paths. We now consider the following two cases.

(a) $x_{p-1} = y_{q-1}$. Let P_1' be the subpath of P_1 from x_p to x_s and let P_2' be the subpath of P_2 from y_q to y_t. Then, the paths P_1' and P_2' along with the edge m_d form a blossom C. Let P_1'' be the subpath of P_1 from x_1 to x_p. By our assumption regarding y_q, P_1'' is disjoint from P_2'. Since P_1'' is an odd M-path from x_1 to x_p disjoint from the blossom C, we have a flower with x_1 as the root. However, $x_1 \in R$, which implies that there is an R flower in G, a contradiction to our assumption.

(b) $x_{p+1} = y_{q-1}$. Let P_1' be the subpath of P_1 from x_1 to x_p and let P_2' be the subpath of P_2 from y_1 to y_{q-1}. Then, the paths P_1' and P_2' are even M-paths from R to the two endpoints of the matched edge (y_q, y_{q-1}). Since we know that the edge (y_{q-1}, y_q) is distinct from the edge m_d, it must be the case that $|P_1'| + |P_2'| < |P_1| + |P_2|$, a contradiction to our choice of m_d.

This concludes the proof that S is a minimum vertex cover of G containing $L \cup R$ and hence the proof of the lemma. $\square$

Since a perfect matching does not leave any vertex unsaturated, we have the following special case of the above lemma which will be a crucial component of the next chapter.

Lemma 4.2.5. *Let $G = (A, B, E)$ be a König graph with a perfect matching M, let $L \subseteq A$ and $R \subseteq B$, be vertex subsets.Then, G has a minimum vertex cover containing $L \cup R$ if and only if G contains neither an odd M-path from $L \cup R$ to $L \cup R$ nor an R-flower.*

4.3 Computing extendable vertex covers

In this section, we show that we can compute a minimum vertex cover containing the annotated vertices of a König graph (if one exists) in polynomial time. We begin with the following definition.

Definition 4.3.1. *Consider a König graph $G = (A, B, E)$ a maximum matching M of G and let U be the set of vertices left unsaturated by M. Let $R(U)$ be the set of vertices in $V \setminus U$ which to which there is an alternating path from a vertex in U. We denote by G/U the graph induced on the set $V \setminus R(U)$.*

Observation 4.3.2. *Consider a König graph $G = (A, B, E)$ a maximum matching M and let U be the set of vertices left unsaturated by M. Then, the graph G/U is a König graph with a perfect matching.*

Proof. Consider the set of alternating trees rooted at the vertices of L where L is the set of neighbors of U. Observe that the union of the vertices in these trees is precisely the set $R(U) \setminus U$. Since by definition, any vertex occurs along with its matching partner in an alternating tree, every vertex in the graph G/U also occurs with its matching partner. Therefore, G/U has a perfect matching. Furthermore, since A is a vertex cover for G, the set $A \setminus R(U)$ is a vertex cover for G/U. Since G/U has a matching saturating $A \setminus R(U)$ across the cut $(A \setminus R(U), B \setminus R(U))$ in G/U, it must be a König graph. □

Lemma 4.3.3. *Consider a König graph $G = (A, B, E)$. Let M be a maximum matching of G and U be the set of vertices left unsaturated by M. Let X be the set of vertices of G to which there is an odd length alternating path from U. Then, for any minimum vertex cover S of G, $X = S \cap R(U)$ and $S \setminus X$ is a minimum vertex cover for G/U.*

Proof. Observe that $X \subseteq A$ since $U \subseteq B$. Let Y be the set of vertices in B whose matching partners are in X. By Observation 4.1.9, for any vertex $y \in Y$, there is a vertex of U labeled *even* in the alternating tree rooted at y. Therefore, since U is not part of any minimum vertex cover of G, neither is Y (Lemma 4.2.3(e)), implying that X is part of every minimum vertex cover of G.

□

Lemma 4.3.4. *Let $G = (A, B, E)$ be a König graph with a perfect matching M, let $L \subseteq A$, $R \subseteq B$ be saturated vertex subsets. If G has a minimum vertex cover containing $L \cup R$, then one such minimum vertex cover can be computed in time $\mathcal{O}(mn)$.*

Proof. Observe that the sets S_1 and S_2 defined in the proof of Lemma 4.2.4 can be computed in time $\mathcal{O}(mn)$. Since we have already shown that $S = S_1 \cup S_2$ is a minimum vertex cover of G, this completes the proof of the lemma. □

Lemma 4.3.5. *Let $G = (A, B, E)$ be a König graph with a maximum matching M, let $L \subseteq A$, $R \subseteq B$ be vertex subsets and let $U \subseteq B$ be the set of vertices left unsaturated by M. If G has a minimum vertex cover containing $L \cup R$, then one such minimum vertex cover can be computed in time $\mathcal{O}(mn)$.*

Proof. We begin by first computing the intersection of the minimum vertex cover with the set $R(U)$ which is part of every minimum vertex cover (Lemma 4.3.3). This can be done in time $\mathcal{O}(mn)$ by constructing alternating trees for every vertex in B. Now, by Lemma 4.3.3, it only remains for us to compute a minimum vertex cover containing $L' = L \setminus R(U)$ and $R' = R \setminus R(U)$ in the graph $G' = G/U$, which is a König graph with a perfect matching. This can be done in time $\mathcal{O}(mn)$ by Lemma 4.3.4, which completes the proof of the lemma. □

4.4 Characterizing König graphs with a unique minimum vertex cover

In this section, we give a complete characterization of König graphs based on the number of minimum vertex covers which they contain.

The following observation follows from Lemma 4.3.3 and implies that it is sufficient for us to restrict our attention to König graphs with perfect matchings.

Observation 4.4.1. *Consider a König graph $G = (A, B, E)$ with a maximum matching M where U is the set of vertices left unsaturated by M. Then, G has a unique minimum vertex cover if and only if the graph G/U, which is a König graph with a perfect matching, has a unique minimum vertex cover.*

Lemma 4.4.2. *Consider a König graph $G = (V, E)$ with a perfect matching M. Then, G has a unique minimum vertex cover if and only if every vertex of G is in the stem of a flower.*

Proof. ($\Rightarrow$) Suppose that G has a unique minimum vertex cover S and there is a vertex $v \in V$ which is not in the stem of a flower. We first consider the case when $v \notin S$. Consider the alternating tree rooted at v, call it T. We now construct a set S' as follows. Initially, set $S' = S \cap (V \setminus V(T))$. Now, for every vertex labeled *even* in T, we add this vertex to S'. Clearly, $|S'| = |S|$ since every matched edge in G contributes exactly 1 vertex to S'. By our assumption, there is a unique minimum vertex cover and hence it must be the case that S' is not a vertex cover. This implies that there is an edge (u, w) not covered by S'. If u and w were both in T, then they must both be labeled *odd*, which implies the presence of a flower with the stem containing v (by Observation 4.1.9). Similarly, if neither u nor w were in T, then the set $S \cap (V \setminus V(T))$ will contain either

u or w since S is a vertex cover of G. Hence, we are left with the case when exactly one of u or w is in T. Assume without loss of generality that u is in T and w is not. Since $w \notin S'$, it must be the case that $w \notin S$, implying that $u \in S$. However, $u \notin S'$ implies that u is labeled *odd* in T. This also implies that the edge (u, w) is not a matched edge and hence the path in T from v to u, along with the edge (u, w) implies that there is an even M-path from v to w, a contradiction to Observation 4.1.9. The case when $v \in S$ can be argued similarly by considering the alternating tree rooted at the matching partner of v which is not in S.

($\Leftarrow$) Suppose that every vertex of G is part of the stem of a flower. Consider an arbitrary vertex v and a flower whose stem contains v. We claim that if the parity of the shortest path along the stem from the base to v is even, then v is in every minimum vertex cover of G and if it is odd, then v is in no minimum vertex cover of G. Suppose that v is the base of the flower and not part of some minimum vertex cover, then the two neighbors of v in the blossom are both in the minimum vertex cover, contradicting Lemma 4.2.3(e) since there is an odd M-path between them. Suppose that v is not the base of this flower. By Lemma 4.2.3(a), if a vertex u has an odd M-path to v, then since v is part of every minimum vertex cover, u is not in any minimum vertex cover. Since no vertex at an odd distance along the stem from the base is in any minimum vertex cover, every vertex at an even distance along the stem from the base is part of every minimum vertex cover. This proves our claim and in fact also shows that if every vertex is part of the stem of a flower, then no vertex will be at an even distance (along the stem) from the base of one flower and at an odd distance (along the stem) from the base of another flower. Hence, the presence or absence of a vertex in any minimum vertex cover of G is solely determined by its position in the stem of any flower. Since every vertex in G is in the stem of a flower, the lemma follows. □

Observation 4.4.1 along with Lemma 4.4.2 gives a complete characterization of König graphs with unique a minimum vertex cover.

4.5 Notes

If we restrict ourselves to bipartite graphs with a perfect matching, it is clear that either partition forms a vertex cover. We note that Lemma 4.4.2 is a generalization of this trivial observation. In other words, since the presence of a flower requires the presence of an odd cycle, by Lemma 4.4.2, a bipartite graph with a perfect matching has at least 2 minimum vertex covers.

We conclude with the following corollary of Observation 4.4.1 applied to bipartite graphs, which is very well known.

Corollary 4.5.1. *A bipartite graph $G = (V, E)$ with a maximum matching M has a unique vertex cover if and only if $U = V$ where U is the set of vertices of G reachable from U via alternating paths.*

5 Vertex Cover Parameterized Above Maximum Matching

5.1 Introduction

The standard version of VERTEX COVER, where we are interested in finding a vertex cover of size at most k for the given parameter k was one of the earliest problems that was shown to be FPT [27]. After a long race, the current best algorithm for VERTEX COVER runs in time $\mathcal{O}(1.2738^k + kn)$ [15]. However, when $k < m$, the size of the maximum matching, the standard version of VERTEX COVER is not interesting, as the answer is trivially NO. And if m is comparable to the size of the vertex set itself, for example, when the graph has a perfect matching, then for the cases the problem is interesting, the running time of the standard version is not practical, as k, is quite large in these cases. This motivates the parameterization of VERTEX COVER above the size of the maximum matching.

ABOVE GUARANTEE VERTEX COVER (AGVC) **Parameter:** k

Input: Graph G, a maximum matching M for G, positive integer k.

Question: Does G have a vertex cover of size at most $|M| + k$?

Prior to the work presented in this chapter, the only known parameterized algorithm for ABOVE GUARANTEE VERTEX COVER was the consequence of a parameter preserving reduction from AGVC to the ALMOST 2-SAT problem. In ALMOST 2-SAT, we are given a 2-SAT formula ϕ, a positive integer k and the objective is to check whether there exists a set of at most k clauses whose deletion from ϕ makes the resulting formula satisfiable. The ALMOST 2-SAT problem was introduced in [72] and a decade later it was shown by Razgon and Barry O'Sullivan [90] to have an $\mathcal{O}^*(15^k)$ time algorithm, thereby proving fixed-parameter tractability of the problem when k is the parameter. The ALMOST 2-SAT problem is a fundamental problem in the context of designing parameterized algorithms. This is evident from the fact that there is a polynomial time parameter preserving reduction from problems like ODD CYCLE TRANSVERSAL [62] and ABOVE GUARANTEE VERTEX COVER [81] to it. An FPT algorithm for ALMOST 2-SAT led to FPT algorithms for several problems, including AGVC and KÖNIG VERTEX DELETION [81]. In recent times this has been used as a subroutine in obtaining a parameterized approximation as well as an FPT algorithm for MULTICUT [78, 79]. Later chapters in the thesis will also demonstrate the utility of having an FPT algorithm for ALMOST 2-SAT. Therefore, a strong motivation for obtaining a faster FPT algorithm for ABOVE GUARANTEE VERTEX COVER is that it also implies a faster FPT algorithm for ALMOST 2-SAT and a number of other problems which are summarized in Fig. 5.1.

In this chapter, we present an improved FPT algorithm for AGVC by using our structural results described in the previous chapter and the important separator template.

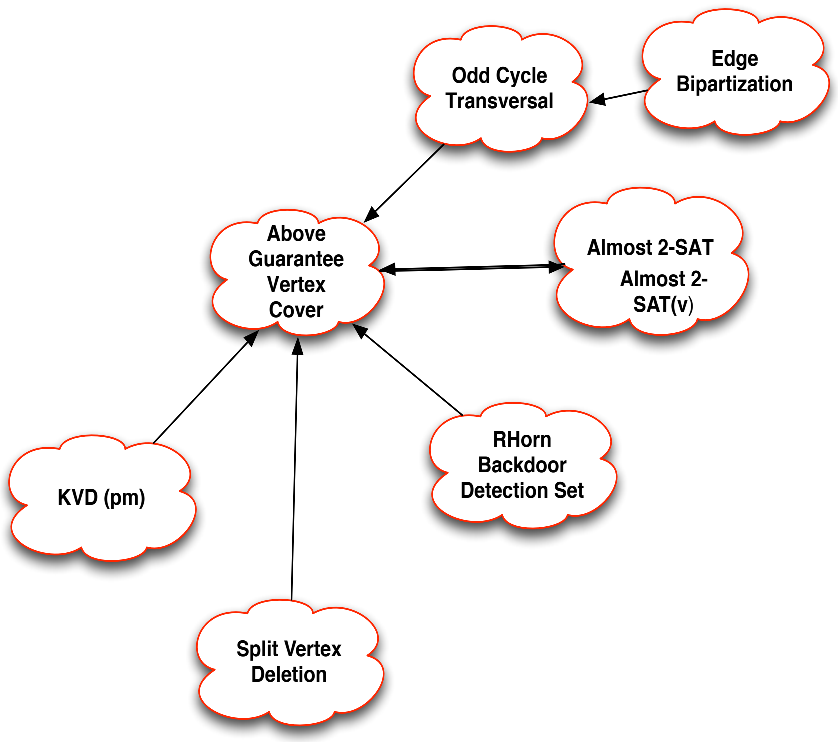

Figure 5.1: Some of the many problems which have a parameter preserving reduction to AGVC. Out of these reductions, the reduction from KVD$_{pm}$ takes the parameter from k to $\frac{k}{2}$ while the rest are all parameter preserving reductions.

5.2 Algorithm for AGVC

We first apply the following lemma that allows us to assume that the input graph has a perfect matching.

Lemma 5.2.1. [81, LEMMA 5] *If* $(G = (V, E), M, k)$ *is an instance of* ABOVE GUARANTEE VERTEX COVER *and* G *is a graph without a perfect matching, then in time* $\mathcal{O}(m\sqrt{n})$, *we can obtain an instance* (G', M', k) *such that* G' *has a perfect matching* M' *and* (G, M, k) *is a* YES *instance of* ABOVE GUARANTEE VERTEX COVER *if and only if* (G', M, k) *is a* YES *instance of* ABOVE GUARANTEE VERTEX COVER*where*

$n = |V|$ *and* $m = |E|$.

Due to Lemma 5.2.1, we assume that in our input instance (G, M, k), the matching M is a perfect matching of G. We now describe the iterative compression step, which is central to our algorithm, in detail.

Iterative Compression for AGVC**.** Given an instance $(G = (V, E), M, k)$ of Above Guarantee Vertex Cover let $M = \{m_1, \ldots, m_{n/2}\}$ be a perfect matching for G where $n = |V|$. Define $M_i = \{m_1, \ldots, m_i\}$, $m_i = (a_i, b_i)$, and $G_i = G[V(M_i)]$, $1 \leq i \leq \frac{n}{2}$. We iterate through the instances (G_i, M_i, k) starting from $i = k + 1$ and for the i^{th} instance, with the help of a *known* solution S_i of size at most $|M_i| + k + 1$ we try to find a solution $\hat{S}_i$ of size at most $|M_i| + k$. Formally, the compression problem we address is the following.

AGVC Compression (AGVCC) **Parameter:** k

Input: (G, M, S, k) where $(G = (V, E), M, k)$ is an instance of AGVC and S a vertex cover of G of size at most $|M| + k + 1$.

Question: Does G have a vertex cover $\hat{S}$ of size at most $|M| + k$?

We reduce the AGVC problem to $\frac{n}{2} - k$ instances of the AGVCC problem as follows. Let $I_i = (G_i, M_i, S_i, k)$ be the i^{th} instance. Clearly, the set $V(M_{k+1})$ is a vertex cover of size at most $|M_{k+1}| + k + 1$ for the instance I_{k+1}. It is also easy to see that if $\hat{S}_{i-1}$ is a vertex cover of size at most $|M_{i-1}| + k$ for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup V(m_i)$ is a vertex cover of size at most $|M_i| + k + 1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+1}, M_{k+1}, S_{k+1}, k)$ where $S_{k+1} = V(M_{k+1})$ and look for a vertex cover of size at most $|M_{k+1}| + k$ for this instance. If we find such a vertex cover $\hat{S}_{k+1}$, we set $S_{k+2} = \hat{S}_{k+1} \cup V(m_{k+2})$ and look for a vertex cover of size at most $|M_{k+2}| + k$ for the instance I_{k+2} and so on. If, during any iteration, the corresponding instance does not have a vertex cover of the required size, it implies that the original instance is also a NO instance. Finally the solution for the original input instance will be $\hat{S}_{\frac{n}{2}}$. Since there can be at most $\frac{n}{2}$ iterations, the total time taken is bounded by $\frac{n}{2}$ times the time required to solve the AGVCC problem.

Our algorithm for AGVCC is as follows. Let the input instance be $I = (G = (V, E), S, M, k)$. Let M' be the edges in M which have both vertices in S. Note that $|M'| \leq k + 1$. Then, $G \setminus V(M')$ is a König graph and by Lemma 4.2.1 has a partition (A, B) such that A is a minimum vertex cover and there is a matching saturating A across the cut (A, B), which in this case is $M \setminus M'$. We guess a subset $Y \subseteq M'$ with the intention of picking both vertices of these edges in our solution (see Fig. 5.2). For the remaining edges of M', exactly one vertex from each edge will be part of our eventual solution. For each edge of $M' \setminus Y$, we guess the vertex which is not part of our eventual solution. Let T be the set of vertices guessed in this way to be disjoint from the solution. Define $L = A \cap N_G(T)$ and $R = B \cap N_G(T)$. Clearly our guess forces $L \cup R$ to be in the solution (see Fig. 5.3). We have thus reduced this problem to the problem of checking if the instance $(G[V(M \setminus M')], A, M \setminus M', k - |M'|)$ has a vertex cover of size at most $|M \setminus M'| + k - |M'|$ which contains L and R. We formally define this annotated variant as follows.

ANNOTATED AGVC (A-AGVC) **Parameter:** k

Input: $(G = (A, B, E), M, L, R, k)$, where $(G = (A, B, E), M, k)$ is an instance of AGVC, $L \subseteq A$ and $R \subseteq B$.

Question: Does G have a vertex cover of size at most $|M| + k$ containing $L \cup R$?

Our main result is the following lemma.

Lemma 5.2.2. *There is an algorithm for* A-AGVC *which runs in time $\mathcal{O}(4^k kmn)$ on an input (G, M, L, R, k), where n is the number of vertices in G and m is the number*

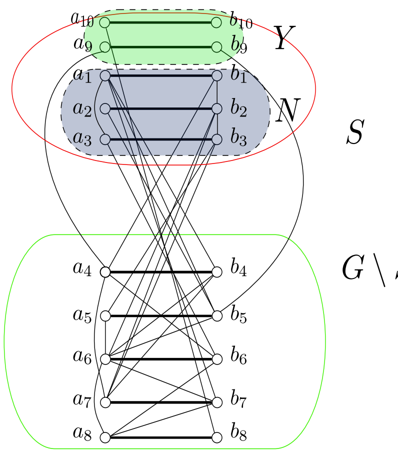

Figure 5.2: An illustration of the partition of S into sets Y and N where Y is part of the solution we want to find and N is disjoint from the solution we want to find.

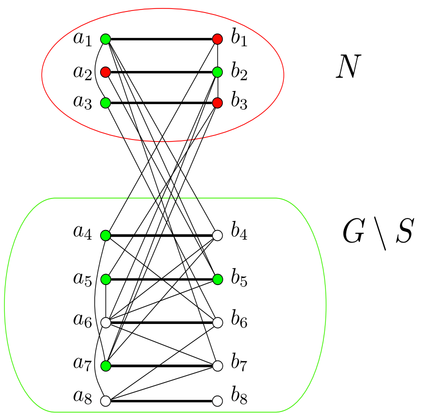

Figure 5.3: An illustration of a partition of N where the red vertices are the vertices in T and the neighbors of T in $G \setminus V(N)$ are forced to be in the solution we are looking for.

of edges in G.

Given Lemma 5.2.2 the running time of our algorithm for AGVCC is bounded as follows. For every $0 \leq i \leq k$, for every i sized subset Y, for every guess of T, we run the algorithm for A-AGVC with parameter $k - i$. For each i, there are $\binom{k+1}{i}$ subsets of M' of size i, and for every choice of Y of size i, there are 2^{k+1-i} choices for T and for every choice of T, running the algorithm for A-AGVC given by Lemma 5.2.2 takes time $\mathcal{O}(4^{k-i}kmn)$. Hence, the running time of our algorithm for AGVCC is bounded by $\mathcal{O}(\Sigma_{i=0}^{k}\binom{k+1}{i}2^{k+1-i}4^{k-i}kmn) = \mathcal{O}(9^k kmn)$ and hence our algorithm for ABOVE GUARANTEE VERTEX COVER runs in time $\mathcal{O}(9^k kmn^2)$. Thus we have the following theorem.

Theorem 5.2.3. AGVC *can be solved in time $\mathcal{O}(9^k kmn^2)$.*

The rest of the chapter is devoted to presenting a proof of Lemma 5.2.2

5.3 Phase 1

In this section, we describe how to apply Phase 1 of the important separator template. In order to do so, we use Lemma 4.2.5 to model the A-AGVC problem as a problem of eliminating certain paths in a directed graph.

Note that, given an instance $(G = (A, B, E), M, L, R, k)$ of A-AGVC, in order to find a minimum vertex cover containing $L \cup R$, it is *sufficient* to find the set M' of matched edges which have both end points in this minimum vertex cover. This follows from the fact that the graph $G \setminus V(M')$ is then a König graph and has a minimum vertex cover that contains $(L \cup R) \setminus V(M')$. Thus, by Lemma 4.3.5, a minimum vertex cover of $G \setminus V(M')$ containing $(L \cup R) \setminus V(M')$ can be computed in polynomial time. Hence, in the rest of the chapter, whenever we talk about *a solution S for an instance of* A-AGVC*, we mean the set of edges of M which have both endpoints in the minimum vertex cover under consideration.*

Definition 5.3.1. *Given an instance $(G = (A, B, E), M, L, R, k)$ of* A-AGVC *we construct a directed graph D(G) corresponding to this instance as follows. Remove all the edges in $G[A]$, orient all the edges of M from A to B and the remaining edges from B to A.*

An immediate observation to this is the following.

Observation 5.3.2. *There is a path from $u \in A$ to $v \in B$ in D(G) if and only if there is an odd M-path from u to v in G. Furthermore, for any set S of arcs in D(G) which correspond to edges of M in G, there is a path from $u \in A$ to $v \in B$ in D(G)$\setminus S$ if and only if there is an odd M-path from u to v in $G \setminus V(S)$.*

Proof. The first statement of the observation follows from the definition of D(G). For the proof of the second statement, suppose that there is a path from $u \in A$ to $v \in B$ in

D(G)$\setminus S$. Since the only out-neighbor of u in D(G) is the matching partner of u and the only in-neighbor of v in D(G) is the matching partner of v, neither u nor v is in the set $V(S)$. Therefore, the u to v path in D(G)$\setminus S$ clearly corresponds to a u to v odd M-path in $G \setminus V(S)$. The converse direction of the second statement simply follows from the definition of D(G). $\square$

Even though the edges of D(G) are directed (and henceforth will be called arcs), they originate in G and have a fixed direction. Hence we will occasionally use the same set of edges/arcs in both the undirected and directed sense. For example we may say that a set S of edges of G is both a solution for the corresponding instance (undirected) and an arc separator in the graph D(G) (directed).

The following lemma gives the first part of Phase 1, that is we show that there is a solution which corresponds to an X-Y separator for some sets X and Y.

Lemma 5.3.3. *Given an instance* $(G = (A, B, E), M, L, R, k)$ *of* A-AGVC*, suppose that there is an* L-R *path in D(G) and let* S *be a solution to this instance. Then,* S *is an* L-R *arc separator in D(G).*

Proof. Since S is a solution, the graph $G \setminus V(S)$ is a König graph with a minimum vertex cover containing $(L \cup R) \setminus V(S)$. By Observation 5.3.2, we know that if there is an L-R path in D(G)$\setminus S$, then there is an odd M-path from L to R in $G \setminus V(S)$ and by Lemma 4.2.5, this implies that there is no minimum vertex cover for $G \setminus V(S)$ containing $(L \cup R) \setminus V(S)$, a contradiction. $\square$

The next lemma is used to handle the case when the instance does not have odd M-paths from L to R.

Lemma 5.3.4. *Let* $(G = (A, B, E), M, L, R, k)$ *be an instance of* A-AGVC *such that there are no odd* M*-paths from* L *to* R *in* G*. If there is an odd* M*-path* P *from* R *to* R

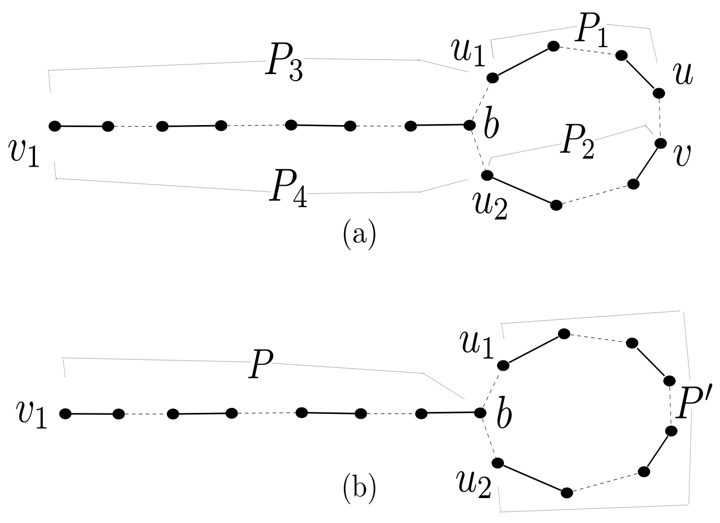

Figure 5.4: An illustration of the two sub cases for the flower in Lemma 5.3.4. (a) $u_1, u_2 \in B$. (b) $u_1 \in A$, $u_2 \in B$.

(or an R-flower $\mathcal{P}$), then there is an edge (u, v) such that $u, v \in A \setminus L$ and there is an odd M-path from u to R and an odd M-path from v to R such that any edge of M which occurs in P (respectively in $\mathcal{P}$) occurs in one of these two odd M-paths. Moreover, this edge can be found in time $\mathcal{O}(mn)$.

Proof. Suppose $P = v_1, \ldots, v_t$ is an R to R odd M-path. Since $v_1, v_t \in R$, $v_1, v_t \in B$ and by Lemma 4.2.3(f) there is an edge (u, v) such that $u, v \in A$ and there are odd M-paths from u and v to v_1 and v_t respectively, which are odd M-paths from u to R and v to R respectively. Furthermore, any edge of M present in P occurs in one of these two odd M-paths.

Suppose that $\mathcal{P}$ is a flower with root $v_1 \in R$ and base b. Let u_1 and u_2 be the neighbors of b in the blossom. We know by Lemma 4.2.3(c) that $b \in A$ and by Lemma 4.2.3(d) that at least one of u_1 and u_2 is in B. We first consider the case when $u_1, u_2 \in B$. Applying Lemma 4.2.3(f) on the blossom path from u_1 to u_2 (see Fig. 5.4) we know that there is an edge (u, v) such that $u, v \in A$ and there are odd M-paths P_1 and P_2 from

u_1 to u and u_2 to v respectively which lie inside the blossom path. Since P_1 and P_2 lie entirely within this blossom path, they do not intersect the stem of the flower. We also know by the definition of flowers that there are even M-paths P_3 and P_4 from the root to u_1 and u_2 respectively. Hence, $Rev(P_1 + P_3)$ and $Rev(P_2 + P_4)$ are odd M-paths from u and v respectively to R and every edge of M present in the stem of $\mathcal{P}$ occurs in P_3 and P_4 and every edge of M present in the blossom of $\mathcal{P}$ occurs in P_1 or P_2.

We now consider the case when exactly one of u_1 and u_2, is in A. Suppose that $u_1 \in A$. We claim that there is an odd M-path from u_1 to R and one from b to R. By the definition of a flower, the stem, say P is an odd M-path from b to R. Let P' be the blossom path from u_1 to u_2. Observe that $P' + (u_2, b) + Rev(P)$ is indeed an odd M-path from u_1 to R and it contains every edge of M which is present in $\mathcal{P}$.

In either of these cases, we have found an edge (u, v) such that $u, v \in A$ and there are odd M-paths from both these vertices to R. Since we have assumed that there are no L to R odd M-paths, $u, v \in A \setminus L$. That we can find this edge in time $\mathcal{O}(mn)$ follows from Observation 4.1.10.

□

Finally we come to the last part of Phase 1, where we show that the problem is solvable in polynomial time if none of the forbidden structures exist. This follows from Lemma 4.2.5, which states that if neither of the forbidden structures exist, then there is indeed a minimum vertex cover for G containing $L \cup R$ and by Lemma 4.3.5, which gives a polynomial time algorithm to compute such a vertex cover.

Summarizing this section, we have shown that either there is a solution for the given instance which is an L-R arc separator in D(G) (Lemma 5.3.3), or there is an edge in G on which we can branch and make progress (Lemma 5.3.4), or the problem is solvable in polynomial time (Lemma 4.3.5). This completes Phase 1 of the important separator template.

5.4 Phase 2

In this section, we demonstrate that Phase 2 of the important separator template is applicable by showing the existence of a **dominating set** of bounded size for the set of L-R arc separators in D(G).

Lemma 5.4.1. *Given an instance* $(G = (A, B, E), M, L, R, k)$ *of* A-AGVC*, any important L-R arc separator in D(G) comprises precisely arcs corresponding to some subset of M.*

Proof. Let X be an important L-R arc separator in D(G). Suppose there is an arc $e = (b_j, a_i) \in X$ such that $e \notin M$. The minimality of X implies that b_j and a_i are reachable from L in D(G) but only b_j is reachable from L in D(G)$\setminus X$. Now, consider the set $X' = (X \setminus e) \cup (a_i, b_i)$. Clearly $|X'| \leq |X|$. Now, X' is also a minimal L-R arc separator in D(G) since any L-R path containing the arc e also contains the arc (a_i, b_i). Now, a_i is reachable from L in D(G)$\setminus X'$ and all vertices reachable from L in D(G)$\setminus X$ are also reachable from L in D(G)$\setminus X'$. Hence the set of vertices reachable from L in D(G)$\setminus X'$ is a strict superset of the set of vertices reachable from L in D(G)$\setminus X$. This contradicts our assumption that X was an important L-R arc separator. □

Lemma 5.4.2. *Let* $(G = (A, B, E), M, L, R, k)$ *be an instance of* A-AGVC*. If* $(G = (A, B, E), M, L, R, k)$ *is a* YES *instance, then it has a solution which contains an important L-R arc separator in D(G).*

Proof. Let S be a solution for the given instance. By Lemma 7.4.3, we have that S is an L-R arc separator in D(G). Let S_{LR} be a minimal subset of S such that the graph D(G)$\setminus S_{LR}$ does not contain a path from L to R.

Let K be the set of vertices reachable from L in D(G)$\setminus S_{LR}$. If S_{LR} is an important L-R arc separator, we are done since S itself is a solution which satisfies our claim. If this were not the case, then there is an important L-R arc separator S'_{LR} such that $|S'_{LR}| \leq |S_{LR}|$ and $K' \supset K$ where K' is the set of vertices reachable from L in D(G)$\setminus S'_{LR}$. Consider the set $\hat{S} = (S \setminus S_{LR}) \cup S'_{LR}$. We claim that $\hat{S}$ is a solution which satisfies our claim. Clearly, $|\hat{S}| \leq |S|$ and $\hat{S}$ contains an important L-R arc separator in D(G). By Lemma 5.4.1, $S'_{LR} \subseteq M$ and hence $\hat{S} \subseteq M$. Therefore, it remains to prove that $\hat{S}$ is indeed a solution to the given instance.

If $\hat{S}$ were not a solution, then $\hat{G} = G \setminus V(\hat{S})$ is a König graph that does not have a minimum vertex cover containing $(L \cup R) \setminus V(\hat{S})$. Lemma 4.2.5 implies that there is either an odd M-path from $L \cup R$ to $L \cup R$ or an R-flower in $\hat{G}$. Since $\hat{S}$ is an L-R arc separator in D(G), $\hat{G}$ does not have odd M-paths from L to R. Therefore there must be either an M-path from R to R or an R-flower in $\hat{G}$. Let this odd M-path (or R-flower) be $\mathcal{P}$. Note that in either case, $\mathcal{P}$ must contain an edge in $S'' = S_{LR} \setminus S'_{LR}$.

By Lemma 5.3.4, we know that there is an edge (u, v) such that $u, v \in A$ and there is an odd M-path from u to R and from v to R in $\hat{G}$, where $u, v \in A$. By Observation 5.3.2, this implies a path from u to R, say P_1 and a path from v to R, say P_2 in D($\hat{G}$). By Lemma 5.3.4, we also know that every edge of M in $\mathcal{P}$ is present in P_1 or P_2. Suppose the edge $m_j = (a_j, b_j)$ be an edge which is present in $S_{LR} \setminus S'_{LR}$ and $\mathcal{P}$. Therefore, m_j is present in either P_1 or P_2. We assume without loss of generality that m_j is present in P_1. Since S'_{LR} dominates S_{LR} with respect to L, it must be the case that a_j is reachable from L in D(G)$\setminus S'_{LR}$ via a path, say P_3. Therefore, combining the path P_3 with the path P_1, we have a path from L to R in D(G) which is disjoint from S'_{LR}, a contradiction. This concludes the proof of the lemma. □

Lemma 5.4.3. *Let* $(G = (A, B, E), M, L, R, k)$ *be an instance of* A-AGVC *and let*

D =D(G)= (V, A) be defined as above. Then the number of important L-R arc separators of size at most k in the graph D is bounded by 4^k.

Proof. Given $D, L, R, k \geq 0$ we define a measure $\mu_a(D, L, R, k) = 2k - \lambda_D(L, R)$. We prove by induction on $\mu_a(D, L, R, K)$ that there are at most $2^{\mu_a(D,L,R,k)}$ important L-R arc separators of size at most k. For the base case, if $2k - \lambda_D(L, R) < k$ then $\lambda_D(L, R) > k$ and hence the number of important L-R arc separators of size at most k is 0. If $\lambda_D(L, R) = 0$, it means that there is no path from L to R and hence the empty set is the only important L-R arc separator. Consider $D, L, R, k \geq 0$ such that $\mu_a = \mu_a(D, L, R, k) \geq k, \lambda_D(L, R) > 0$ and assume that the statement of the Lemma holds for all D', L', R', k' where $\mu_a(D', L', R', k') < \mu_a$.

By Lemma 3.2.26, there is a unique important L-R arc separator S^* of size $\lambda_D(L, R)$. Since we have assumed $\lambda_D(L, R)$ to be positive, S^* is non empty. Consider an arc $e = (u, v) \in S^*$. By Lemma 5.4.1, there is some i such that $u = a_i$ and $v = b_i$. Any important L-R arc separator S either contains e or does not contain e. For any important L-R arc separator S which contains e, $S \setminus \{e\}$ is an important L-R arc separator in $D \setminus \{e\}$ by Lemma 3.2.30(2). Hence the number of important L-R arc separators of size at most k in D which contain e, is at most the number of important L-R arc separators of size at most $k - 1$ in $D \setminus \{e\}$. Observe that $\lambda_{D \setminus \{e\}}(L, R) = \lambda_D(L, R) - 1$ which implies that $\mu_a(D \setminus \{e\}, L, R, k - 1) < \mu_a$ and by the induction hypothesis, the number of important L-R arc separators of size at most $k - 1$ in $D \setminus \{e\}$ is bounded by $2^{\mu_a - 1}$ which is also a bound on the number of important L-R arc separators of size at most k in D which contain e.

Now let S be an important L-R arc separator of size at most k which does not contain e. By Lemma 3.2.29 we know that $K_S \supseteq K_{S^*}$ and by the minimality of S^*, $a_i \in K_{S^*}$ and since $K_S \supseteq K_{S^*}$, a_i is in K_S. But $e \notin S$, which implies that b_i is in K_S which implies that $K_S \supseteq K_{S^*} \cup \{b_i\}$. But by Lemma 5.4.1, no other edge incident on b_i can be in S. Hence the vertices in $\delta^+(b_i)$ are also reachable from L in $D \setminus S$. We now set $X = K_{S^*} \cup \delta^+(b_i)$ and by Lemma 3.2.30(3) we know that S is also an important X-R arc separator. We set $L' = A \cap X$. Since there cannot be paths from R to R in D(G), any X-R separator is also an L'-R separator. Thus a bound on the number of important L'-R arc separators of size at most k is also a bound on the number of important L-R arc separators of size at most k which do not contain the arc e. First note that $\lambda_D(L', R) > \lambda_D(L, R)$ since otherwise we would have an L-R arc separator S' of size at most S^* such that $K_{S'} \supset K_{S^*}$. Now, $\mu_a(D, L', R, k) < \mu_a$ and by the induction hypothesis, the number of important L'-R arc separators of size at most k is bounded by $2^{\mu_a - 1}$.

Summing up the upper bounds in both cases, the number of important L-R arc separators of size at most k is bounded by $2.2^{\mu_a-1} = 2^{\mu_a} \leq 2^{2k}$. $\square$

We have thus shown that the set of important L-R arc separators of size at most k is indeed a **dominating set** for the set of L-R arc separators of size at most k with respect to this problem (Lemma 5.4.2) and that the size of the **dominating set** is bounded by 4^k (Lemma 5.4.3). Therefore, there is a solution which intersects the set of $4^k k$ vertices in the union of the vertices in all important separators of size at most k. This completes Phase 2 of the important separator template.

5.5 Phase 3

In this section, we show that the **dominating set**, that is, the set of important L-R arc separators defined in the previous section can be computed in FPT time. The proof is the same as that seen in Chapter 3. However, since the analysis of our final algorithm will require us to take a closer look at how this particular algorithm is interlinked with the rest of the steps, we have presented the algorithm to compute the set of important arc separators in terms of the problem at hand.

Lemma 5.5.1. *The set of important L-R arc separators of size at most k can be enumerated in time $\mathcal{O}(4^k k(m+n))$.*

Proof. The algorithm for enumerating the important L-R arc separators of size at most k follows from the proof of Lemma 5.4.3. The algorithm first computes the unique smallest important L-R separator S^* using the algorithm described in Lemma 3.2.28, selects an arc $e \in S^*$ and recursively enumerates all important L-R arc separators which contain e and those which do not. It follows from the proof of Lemma 5.4.3 that this algorithm runs in time $\mathcal{O}(4^k k(m+n))$. □

Thus, Lemma 5.5.1 gives an algorithm running in time $\mathcal{O}(4^k k(m+n))$ to compute the **dominating set**. This completes Phase 3.

5.6 Phase 4

Finally, we combine the first three phases (summarized in Fig. 9.7) to obtain our algorithm for A-AGVC.

We are now ready to prove Lemma 5.2.2 by describing an algorithm (Algorithm. 5.6.1) for A-AGVC. In order to make the analysis of our algorithm simpler (and stronger), we embed the algorithm for enumerating important separators (Lemma 5.4.3) into our algorithm for A-AGVC.

Correctness. The Correctness of Step 1 is obvious. In Steps 6 and 8 we are merely guessing the vertex which covers the edge (u, v), while Step 11 is correct due to Lemma 4.2.5.

```
Input  : An instance $(G, M, L, R, k)$ of A-AGVC
Output: A solution of size at most $k$ for the instance $(G, M, L, R, k)$ if it exists
        and NOotherwise
1  if $k < 0$ then return NO
2  Compute a mimimum size $L$-$R$ arc separator $S$ in the directed graph D($G$)
3  if $|S| = 0$ then
4      if there an odd $M$-path from $R$ to $R$ or an $R$-flower then
5          compute the edge $e = (u, v)$ given by Lemma 5.3.4
6          $S_1 \leftarrow Solve - AAGVC(G, M, L \cup \{u\}, R, k)$
7          if $S_1$ is not NO then return $S_1$
8          $S_2 \leftarrow Solve - AAGVC(G, L \cup \{v\}, R, k)$
9          return $S_2$
10     end
11     else return $\phi$
12 end
13 if $|S| > k$ then return NO
14 else Compute the unique minimum size important $L$-$R$ separator $S^*$ in D($G$)
   (Lemma 3.2.28) and select an arc $e = (w, z) \in S^*$
15 $S_3 \leftarrow Solve - AAGVC(G \setminus \{e\}, M \setminus \{e\}, L, R, k - 1)$
16 if $S_3$ is not NO then return $S_3 \cup \{e\}$
17 $S_4 \leftarrow Solve - AAGVC(G, M, A \cap (\delta^+_{D(G)}(z) \cup K_{S^*}), R, k)$
18 return $S_4$
```

Algorithm 5.6.1: Algorithm $Solve - AAGVC$ for A-AGVC

Phase 1.
(a) If there is an odd M-path from L to R in G, then the solution contains an L-R arc separator in D(G) (Lemma 5.3.3).
(b) If there are no odd M-paths from L to R, but there is either an odd M-path from R to R or an R-flower, then there exists a branch-able edge (u, v) between two vertices in A which can be computed in time $\mathcal{O}(mn)$ (Lemma 5.3.4)
(c) If neither of these two cases occur, then the graph already has a minimum vertex cover containing $L \cup R$ and this can be computed in time $\mathcal{O}(mn)$ (Lemma 4.2.5).

Phase 2. If there is a solution containing an L-R arc separator in D(G), there is one which contains an important L-R arc separator in D(G) (Lemma 5.4.2). The number of important L-R arc separators of size at most k is at most 4^k (Lemma 5.4.3) and hence the number of vertices in their union is at most $4^k k$.

Phase 3. The set of important L-R arc separators of size at most k in D(G) can be enumerated in time $\mathcal{O}(4^k kmn)$ (Lemma 5.5.1).

Step 13 is correct because the size of the minimum L-R separator in D(G) is a lower bound on the solution size. Steps 15 and 17 are part of enumerating the important L-R arc separators as seen in Lemma 5.4.3. Since we have shown in Lemma 5.4.2 that if there is a solution, there is one which contains an important L-R separator in D(G), these steps are also correct.

Running Time. In order to analyze the algorithm, we define the search tree $\mathbb{T}(G, M, L, R, k)$ resulting from a call to $Solve - AAGVC(G, M, L, R, k)$ inductively as follows. The tree $\mathbb{T}(G, M, L, R, k)$ is a rooted tree whose root node corresponds to the instance (G, M, L, R, k). If $Solve - AAGVC(G, M, L, R, k)$ does not make a recursive call, then (G, M, L, R, k) is said to be the only node of this tree. If it does make recursive calls, then the children of (G, M, L, R, k) correspond to the instances given as input to the recursive calls made inside the current procedure call. The subtree rooted at a child node (G', M', L', R', k') is the search tree $\mathbb{T}(G', M', L', R', k')$.

Given an instance $I = (G, M, L, R, k)$, we prove by induction on $\mu(I) = 2k - \lambda_{D(G)}(L, R)$ that the number of leaves of the tree $\mathbb{T}(I)$ is bounded by $max\{2^{2\mu(I)}, 1\}$. In the base case, if $\mu(I) < k$, then $\lambda(L, R) > k$ in which case the number of leaves is 1. Assume that $\mu(I) \geq k$ and our claim holds for all instances I' such that $\mu(I') < \mu(I)$.

Suppose $\lambda(L, R) = 0$. In this case, the children I_1 and I_2 of this node correspond to the recursive calls made in Steps 6 and 8. By Lemma 5.3.4 there are odd M-paths from u to R and from v to R. Hence, $\lambda(L \cup \{u\}, R) > 0$ and $\lambda(L \cup \{v\}, R) > 0$. This implies that $\mu(I_1), \mu(I_2) < \mu(I)$. By the induction hypothesis, the number of leaves in the search trees rooted at I_1 and I_2 are at most $2^{\mu(I_1)}$ and $2^{\mu(I_2)}$ respectively. Hence the number of leaves in the search tree rooted at I is at most $2.2^{\mu(I)-1} = 2^{\mu(I)}$.

Suppose $\lambda(L, R) > 0$. In this case, the children I_1 and I_2 of this node correspond to the recursive calls made in Steps 15 and 17. But in these two cases, as seen in the proof of Lemma 5.4.3, $\mu(I_1), \mu(I_2) < \mu(I)$ and hence by applying the induction hypothesis on the two child nodes and summing up the number of leaves in the subtrees rooted at each, we can bound the number of leaves in the subtree of I by $2^{\mu(I)}$.

Therefore, the number of leaves of the search tree $\mathbb{T}$ rooted at the input instance $I = (G, M, L, R, k)$ is $2^{\mu(I)} \leq 2^{2k}$. The time spent at any internal node of the search tree is bounded by $\mathcal{O}(k(m + n) + mn)$ since at every node, we either find an M-path from R to R or an R-flower or find a minimum L-R arc separator if one of size at most k exists. Therefore the running time of the algorithm is $\mathcal{O}(4^k kmn)$. This completes the proof of Lemma 5.2.2.

5.7 Corollaries

The following corollaries are obtained using known FPT reductions [88, 81, 44].

Lemma 5.7.1. ALMOST 2-SAT *can be solved in time* $\mathcal{O}(9^k kmn^2)$.

Lemma 5.7.2. KÖNIG VERTEX DELETION *on graphs with a perfect matching can be solved in time* $\mathcal{O}(3^k kmn^2)$.

Lemma 5.7.3. RHORN BACKDOOR DETECTION *can be solved in time* $\mathcal{O}(9^k kmn^2)$.

6

Vertex Cover Parameterized Above LP Optimum

6.1 Introduction

The well known integer linear programming formulation (ILP) for VERTEX COVER is as follows.

ILP FORMULATION OF MINIMUM VERTEX COVER – ILPVC	
Instance:	A graph $G = (V, E)$.
Feasible Solution:	A function $x : V \rightarrow \{0, 1\}$ satisfying edge constraints $x(u) + x(v) \geq 1$ for each edge $(u, v) \in E$.
Goal:	To minimize $w(x) = \Sigma_{u \in V} x(u)$ over all feasible solutions x.

In the standard linear programming relaxation of the above ILP, the constraint $x(v) \in \{0, 1\}$ is replaced with $x(v) \geq 0$, for all $v \in V$. For a graph G, we call this relaxation LPVC(G). Clearly, every integer feasible solution is also a feasible solution to LPVC(G). Clearly the size of a minimum vertex cover of a graph is at least the min-

Problem Name	Previous $f(k)$/Reference	New $f(k)$ in this paper
AGVC	4^k [23]	2.3146^k
ALMOST 2-SAT	4^k [23]	2.3146^k
RHORN-BACKDOOR SET DETECTION	4^k [23, 44]	2.3146^k
KÖNIG VERTEX DELETION	4^k [23, 81]	1.5214^k
SPLIT VERTEX DELETION	5^k [12]	2.3146^k
ODD CYCLE TRANSVERSAL	3^k [91]	2.3146^k
VERTEX COVER PARAM BY OCT	2^k (folklore)	1.5214^k
VERTEX COVER PARAM BY KVD	–	1.5214^k

Table 6.1: The table gives the previous $f(k)$ bound in the running time of various problems and the ones obtained in this paper.

imum value of LPVC for the graph. This allows us to parameterize VERTEX COVER above the minimum value of LPVC for the input graph. Prior to our study, a similar parameterization had recently been studied by Cygan et al. [23] in the context of the MULTIWAY CUT problem, as a consequence of which they obtained an $\mathcal{O}(4^k n^{\mathcal{O}(1)})$ algorithm for VERTEX COVER parameterized above the size of the maximum matching.

In this chapter, we develop a $\mathcal{O}(2.3146^{(k-vc^*(G))} n^{O(1)})$ time branching algorithm for VERTEX COVER ABOVE LP. In an effort to present the key ideas of our algorithm in as clear a way as possible, we first present a simpler and slightly slower algorithm in Section 6.3. This algorithm exhaustively applies a collection of previously known preprocessing steps. If no further preprocessing is possible the algorithm simply selects an arbitrary vertex v and recursively tries to find a vertex cover of size at most k by considering whether v is in the solution or not. While the algorithm is simple, the analysis is more involved as it is not obvious that the measure $k - vc^*(G)$ actually drops in the recursive calls. In order to prove that the measure does drop we string together several known results about the linear programming relaxation of VERTEX COVER, such as the classical Nemhauser-Trotter theorem and properties of “minimum surplus sets”. We find it intriguing that, as our analysis shows, combining well-known reduction rules with naive branching yields fast FPT algorithms for all problems in Figure 10.1. We then show in Section 6.4 that adding several more involved branching rules to our algorithm yields an improved running time of $\mathcal{O}(2.3146^{(k-vc^*(G))}n^{O(1)})$. Using this algorithm we obtain even faster algorithms for the problems in Figure 10.1.

We give a list of problems with their previous best running time and the ones obtained in this paper in Table 6.1. The most notable among them is the new algorithm for ODD CYCLE TRANSVERSAL, the problem of deleting at most k vertices to obtain a bipartite graph. The parameterized complexity of ODD CYCLE TRANSVERSAL was a long standing open problem in the area, and only in 2003 Reed et al. [91] developed an algorithm for the problem running in time $\mathcal{O}^*(3^k)$. However, there has been no further improvement over this algorithm in the last 9 years; though reinterpretations of the algorithm have been published [49, 69].

We also find the algorithm for KÖNIG VERTEX DELETION, the problem of deleting at most k vertices to obtain a König graph very interesting. KÖNIG VERTEX DELETION is a natural variant of the odd cycle transversal problem. In [81] it was shown that given a minimum vertex cover one can solve KÖNIG VERTEX DELETION in polynomial time. In this article we show a relationship between the measure $k - vc^*(G)$ and the minimum number of vertices needed to delete to obtain a König graph. This relationship together with a reduction rule for KÖNIG VERTEX DELETION based on the Nemhauser-Trotter theorem gives an algorithm for the problem with running time $\mathcal{O}^*(1.5214^k)$.

We also note that using our algorithm, we obtain a polynomial time algorithm for VERTEX COVER that, given an input (G, k) returns an equivalent instance $(G' = (V', E'), k')$ such that $k' \leq k$ and $|V(G')| \leq 2k - c\log k$ for any fixed constant c. This is known as a kernel for VERTEX COVER in the literature. We note that this kernel is simpler than another kernel with the same size bound [65].

Organization of the chapter. In Section 6.3, we give a simple branching algorithm and introduce the nature of analysis which will be required for such algorithms. In Section 6.4, we present our main algorithm using much more involved branching rules. In Section 10.4, we give the applications of our result to numerous other parameterized problems.

6.2 Preliminaries

The surplus of an independent set $X \subseteq V$ is defined as $\textbf{surplus}(X) = |N(X)| - |X|$. For a set $\mathcal{A}$ of independent sets of a graph, $\textbf{surplus}(\mathcal{A}) = \min_{X \in \mathcal{A}} \textbf{surplus}(X)$. The surplus of a graph G, **surplus(G)**, is defined to be the minimum surplus over all independent sets in the graph.

By the phrase "an optimum solution to LPVC(G)", we mean a feasible solution with $x(v) \geq 0$ for all $v \in V$ minimizing the objective function $w(x) = \sum_{u \in V} x(u)$. It is well known that for any graph G, there exists an optimum solution to LPVC(G), such that $x(u) \in \{0, \frac{1}{2}, 1\}$ for all $u \in V$ [83]. Such a feasible optimum solution to LPVC(G) is called a half integral solution and can be found in polynomial time [83]. In this chapter we always deal with half integral optimum solutions to LPVC(G). Thus, by default whenever we refer to an *optimum solution* to LPVC(G) we will be referring to a *half integral optimum solution* to LPVC(G). Furthermore, it is also known that the modified LP resulting from forcing certain variables to a value in $\{0, \frac{1}{2}, 1\}$ also has a half integral optimum solution. Let $VC(G)$ be the set of all minimum vertex covers of G and $vc(G)$ denote the size of a minimum vertex cover of G. Let $VC^*(G)$ be the set of all optimal solutions (including non half integral optimal solution) to LPVC(G). By $vc^*(G)$ we denote the value of an optimum solution to LPVC(G). We define $V_i^x = \{u \in V : x(u) = i\}$ for each $i \in \{0, \frac{1}{2}, 1\}$ and define $x \equiv i$, $i \in \{0, \frac{1}{2}, 1\}$, if $x(u) = i$ for every $u \in V$. Clearly, $vc(G) \geq vc^*(G)$ and $vc^*(G) \leq \frac{|V|}{2}$ since $x \equiv \frac{1}{2}$ is always a feasible solution to LPVC(G). We also refer to the $x \equiv \frac{1}{2}$ solution simply as the all $\frac{1}{2}$ solution.

In branching algorithms, we say that a branching step results in a drop of $(p_1, p_2, ..., p_l)$ where p_i, $1 \leq i \leq l$ is an integer, if the measure we use in the analysis drops respectively by $p_1, p_2, ..., p_l$ in the corresponding branches. We also call the vector $(p_1, p_2, \ldots, p_l)$ the branching vector of the step.

6.3 A Simple Algorithm for VERTEX COVER ABOVE LP

In this section, we give a simpler algorithm for VERTEX COVER ABOVE LP. The algorithm has two phases, a preprocessing phase and a branching phase. We first describe the preprocessing steps used in the algorithm and then give a simple description of the algorithm. Finally, we prove its correctness and the bound on the running time of the algorithm.

6.3.1 Preprocessing

We describe three standard preprocessing rules to simplify the input instance. We first state the (known) results which allow for their correctness, and then describe the rules.

Lemma 6.3.1. [84, 87] *For a graph G, in time $\mathcal{O}(m\sqrt{n})$, we can compute an optimal solution x to LPVC(G) where n is the number of vertices in G and m is the number of edges. Furthermore, in time $\mathcal{O}(mn\sqrt{n})$, we can compute an optimal solution x to LPVC(G)such that all $\frac{1}{2}$ is the unique optimal solution to LPVC($G[V^x_{1/2}]$). Furthermore,* **surplus**$(G[V^x_{1/2}]) > 0$.

Lemma 6.3.2. [84] *Let G be a graph and x be an optimal solution to LPVC(G). There is a minimum vertex cover for G which contains all the vertices in V_1^x and none of the vertices in V_0^x.*

Preprocessing Rule 1. *Apply Lemma 6.3.1 to compute an optimal solution x to LPVC(G) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V_{1/2}^x]$). Delete the vertices in $V_0^x \cup V_1^x$ from the graph after including V_1^x in the vertex cover we develop, and reduce k by $|V_1^x|$.*

In the discussions in the rest of the chapter, we say that Preprocessing Rule 1 applies (or is applicable) if all $\frac{1}{2}$ is not the unique solution to LPVC(G) and that it doesn't apply (or is not applicable) if all $\frac{1}{2}$ is the unique solution to LPVC(G).
The soundness/correctness of Preprocessing Rule 1 follows from Lemma 6.3.2. The time required to check if it is applicable and to apply it is $\mathcal{O}(mn\sqrt{n})$. After the application of Preprocessing Rule 1, we know that $x \equiv \frac{1}{2}$ is the unique optimal solution to LPVC of the resulting graph and the graph has a surplus of at least 1.

Lemma 6.3.3. [15, 84] *Let $G(V, E)$ be a graph, and let $S \subseteq V$ be an independent subset such that $\mathbf{surplus}(Y) \geq \mathbf{surplus}(S)$ for every set $Y \subseteq S$. Then there exists a minimum vertex cover for G that contains either all of S or none of S. In particular, if S is an independent set with the minimum surplus, then there exists a minimum vertex cover for G, that contains all of S or none of S.*

The following lemma, which handles without branching, the case when the minimum surplus of the graph is 1, follows from the above lemma.

Lemma 6.3.4. [15, 84] *Let G be a graph, and let $Z \subseteq V(G)$ be an independent set such that $\mathbf{surplus}(Z) = 1$ and for every $Y \subseteq Z$, $\mathbf{surplus}(Y) \geq \mathbf{surplus}(Z)$. Then,*

1. *If the graph induced by $N(Z)$ is not an independent set, then there exists a minimum vertex cover in G that includes all of $N(Z)$ and excludes all of Z.*

2. *If the graph induced by $N(Z)$ is an independent set, let G' be the graph obtained from G by removing $Z \cup N(Z)$ and adding a vertex z, followed by making z adjacent to every vertex $v \in G \setminus (Z \cup N(Z))$ which was adjacent to a vertex in $N(Z)$ (also called* identifying *the vertices of $N(Z)$).Then, G has a vertex cover of size at most k if and only if G' has a vertex cover of size at most $k - |Z|$.*

We now give two preprocessing rules to handle the case when the surplus of the graph is 1.

Preprocessing Rule 2. *If there is a set Z such that* $\textbf{surplus}(Z) = 1$ *and $N(Z)$ is not an independent set, then apply Lemma 6.3.4 to reduce the instance as follows. Include $N(Z)$ in the vertex cover, delete $Z \cup N(Z)$ from the graph, and decrease k by $|N(Z)|$.*

Preprocessing Rule 3. *If there is a set Z such that* $\textbf{surplus}(Z) = 1$ *and $N(Z)$ is an independent set, then apply Lemma 6.3.4 to reduce the instance as follows. Remove Z from the graph, identify the vertices of $N(Z)$, and decrease k by $|Z|$.*

The correctness of Preprocessing Rules 2 and 3 follows from Lemma 6.3.4. The entire preprocessing phase of the algorithm is summarized in Figure 6.1. Recall that each preprocessing rule can be applied only when none of the preceding rules are applicable, and that Preprocessing Rule 1 is applicable if and only if there is a solution to LPVC(G) which does not assign $\frac{1}{2}$ to every vertex. Hence, when Preprocessing Rule 1 does not apply, all $\frac{1}{2}$ is the unique solution for LPVC(G). We now show that we can test whether Preprocessing Rules 2 and 3 are applicable on the current instance in polynomial time.

Lemma 6.3.5. *Given an instance (G, k) of* Vertex Cover Above LP *on which Preprocessing Rule 1 does not apply, we can test if Preprocessing Rule 2 applies on this instance in time $\mathcal{O}(m^2\sqrt{n})$.*

Proof. We first prove the following claim.

Claim 1. *The graph G (in the statement of the lemma) contains a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ is not independent if and only if there is an edge $(u, v) \in E$ such that solving LPVC(G) with $x(u) = x(v) = 1$ results in a solution with value exactly $\frac{1}{2}$ greater than the value of the original LPVC(G).*

Proof. Suppose there is an edge (u, v) such that $w(x') = w(x) + \frac{1}{2}$ where x is the solution to the original LPVC(G) and x' is the solution to LPVC(G) with $x'(u) = x'(v) = 1$ and let $Z = V_0^{x'}$. We claim that the set Z is a set with surplus 1 and that $N(Z)$ is not independent. Since $N(Z)$ contains the vertices u and v, $N(Z)$ is not an independent set. Now, since $x \equiv \frac{1}{2}$ (Preprocessing Rule 1 does not apply), $w(x') = w(x) - \frac{1}{2}|Z| + \frac{1}{2}|N(Z)| = w(x) + \frac{1}{2}$. Hence, $|N(Z)| - |Z| = \mathbf{surplus}(Z) = 1$.

Conversely, suppose that there is a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ contains vertices u and v such that $(u, v) \in E$. Let x' be the assignment which assigns 0 to all vertices in Z, 1 to all vertices in $N(Z)$ and $\frac{1}{2}$ to the rest of the vertices. Clearly, x' is a feasible assignment and $w(x') = |N(Z)| + \frac{1}{2}|V \setminus (Z \cup N(Z))|$. Since Preprocessing Rule 1 does not apply, $w(x') - w(x) = |N(Z)| - \frac{1}{2}(|Z| + |N(Z)|) = \frac{1}{2}(|N(Z)| - |Z|) = \frac{1}{2}$, which proves the converse part of the claim. □

Given the above claim, we check if Preprocessing Rule 2 applies by doing the following for every edge (u, v) in the graph. Set $x(u) = x(v) = 1$ and solve the resulting LP looking for a solution whose optimum value is exactly $\frac{1}{2}$ more than the optimum value of LPVC(G). The time required to check for applicability and to apply the rule is bounded by m times the time to compute an optimum solution to LPVC(G), which is $\mathcal{O}(m^2\sqrt{n})$. $\square$

Lemma 6.3.6. *Given an instance (G,k) of* VERTEX COVER ABOVE LP *on which Preprocessing Rules 1 and 2 do not apply, we can test if Preprocessing Rule 3 applies on this instance in time $\mathcal{O}(mn\sqrt{n})$.*

Proof. We first prove a claim analogous to that proved in the previous lemma.

Claim 2. *The graph G (in the statement of the lemma) contains a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ is independent if and only if there is a vertex $u \in V$ such that solving LPVC(G) with $x(u) = 0$ results in a solution with value exactly $\frac{1}{2}$ greater than the value of the original LPVC(G).*

Proof. Suppose there is a vertex u such that $w(x') = w(x) + \frac{1}{2}$ where x is the solution to the original LPVC(G) and x' is the solution to LPVC(G) with $x'(u) = 0$ and let $Z = V_0^{x'}$. We claim that the set Z is a set with surplus 1 and that $N(Z)$ is independent. Since $x \equiv \frac{1}{2}$ (Preprocessing Rule 1 does not apply), $w(x') = w(x) - \frac{1}{2}|Z| + \frac{1}{2}|N(Z)| = w(x) + \frac{1}{2}$. Hence, $|N(Z)| - |Z| = \mathbf{surplus}(Z) = 1$. Since Preprocessing Rule 2 does not apply, it must be the case that $N(Z)$ is independent.

Conversely, suppose that there is a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ is independent. Let x' be the assignment which assigns 0 to all vertices of Z and 1 to all vertices of $N(Z)$ and $\frac{1}{2}$ to the rest of the vertices. Clearly, x' is a feasible assignment and $w(x') = |N(Z)| + \frac{1}{2}|V \setminus (Z \cup N(Z))|$. Since Preprocessing Rule 1 does not apply, $w(x') - w(x) = |N(Z)| - \frac{1}{2}(|Z| + |N(Z)|) = \frac{1}{2}(|N(Z)| - |Z|) = \frac{1}{2}$. This proves the converse part of the claim with u being any vertex of Z. $\square$

Given the above claim, we check if Preprocessing Rule 3 applies by doing the following for every vertex u in the graph. Set $x(u) = 0$, solve the resulting LP and look for

The rules are applied in the order in which they are presented, that is, any rule is applied only when none of the earlier rules are applicable.

Preprocessing rule 1: Apply Lemma 6.3.1 to compute an optimal solution x to LPVC(G) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V^x_{1/2}]$). Delete the vertices in $V^x_0 \cup V^x_1$ from the graph after including V^x_1 in the vertex cover we develop, and reduce k by $|V^x_1|$.

Preprocessing rule 2:Apply Lemma 6.3.5 to test if there is a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ is not an independent set. If such a set does exist, then we apply Lemma 6.3.4 to reduce the instance as follows. Include $N(Z)$ in the vertex cover, delete $Z \cup N(Z)$ from the graph, and decrease k by $|N(Z)|$.

Preprocessing rule 3: Apply Lemma 6.3.6 to test if there is a set Z such that $\mathbf{surplus}(Z) = 1$ and $N(Z)$ is an independent set. If there is such a set Z then apply Lemma 6.3.4 to reduce the instance as follows. Remove Z from the graph, identify the vertices of $N(Z)$, and decrease k by $|Z|$.

Figure 6.1: Preprocessing Steps

a solution whose optimum value exactly $\frac{1}{2}$ more than the optimum value of $LPVC(G)$. The time required to check for applicability and to apply the rule is bounded by n times the time to compute an optimum solution to LPVC(G), which is $\mathcal{O}(mn\sqrt{n})$. $\square$

Definition 6.3.7. *For a graph G, we denote by $\mathcal{R}(G)$ the graph obtained after applying Preprocessing Rules 1, 2 and 3 exhaustively in this order.*

Strictly speaking $\mathcal{R}(G)$ is not a well defined function since the reduced graph could depend on which sets the reduction rules are applied on, and these sets, in turn, depend on the solution to the LP. To overcome this technicality we let $\mathcal{R}$ be a function not only of the graph G but also of the representation of G in memory. Since our reduction rules are deterministic (and the LP solver we use as a black box is deterministic as well), running the reduction rules on (a specific representation of) G will always result in the same graph, making the function $\mathcal{R}(G)$ well defined. Finally, observe that for any G

the all $\frac{1}{2}$ solution is the unique optimum solution to the LPVC($\mathcal{R}(G)$) and $\mathcal{R}(G)$ has a surplus of at least 2.

6.3.2 Branching

After the preprocessing rules are applied exhaustively, we pick an arbitrary vertex u in the graph and branch on it. In other words, in one branch, we add u into the vertex cover, decrease k by 1, and delete u from the graph, and in the other branch, we add $N(u)$ into the vertex cover, decrease k by $|N(u)|$, and delete $\{u\} \cup N(u)$ from the graph. The correctness of this algorithm follows from the soundness of the preprocessing rules and the fact that the branching is exhaustive.

6.3.3 Analysis

In order to analyze the running time of our algorithm, we define a measure $\mu = \mu(G, k) = k - vc^*(G)$. We first show that our preprocessing rules do not increase this measure. Following this, we will prove a lower bound on the decrease in the measure occurring as a result of the branching, thus allowing us to bound the running time of the algorithm in terms of the measure μ. For each case, we let (G', k') be the instance resulting by the application of the preprocessing rule or branch, and let x' be an optimum solution to LPVC(G').

1. Consider the application of Preprocessing Rule 1. We know that $k' = k - |V_1^x|$. Since $x' \equiv \frac{1}{2}$ is the unique optimum solution to LPVC(G'), and G' comprises precisely the vertices of $V_{1/2}^x$, the value of the optimum solution to LPVC(G') is exactly $|V_1^x|$ less than that of G. Hence, $\mu(G, k) = \mu(G', k')$.
2. We now consider the application of Preprocessing Rule 2 and let V' be the set of vertices in the graph resulting from the application of the rule. We know that

$N(Z)$ was not independent. In this case, $k' = k - |N(Z)|$. We also know that $w(x') = \sum_{u \in V'} x'(u) = w(x) - \frac{1}{2}(|Z| + |N(Z)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(Z)|)$, we get $w(x') = w(x) - |N(Z)| + \frac{1}{2}(|N(Z)| - |Z|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. But, $Z \cup V_0^{x'}$ is an independent set in G, and $N(Z \cup V_0^{x'}) = N(Z) \cup V_1^{x'}$ in G. Since $\mathbf{surplus}(G) \geq 1$, $|N(Z \cup V_0^{x'})| - |Z \cup V_0^{x'}| \geq 1$. Hence, $w(x') = w(x) - |N(Z)| + \frac{1}{2}(|N(Z \cup V_0^{x'})| - |Z \cup V_0^{x'}|) \geq w(x) - |N(Z)| + \frac{1}{2}$. Thus, $\mu(G', k') \leq \mu(G, k) - \frac{1}{2}$.

3. We now consider the application of Preprocessing Rule 3. We know that $N(Z)$ was independent. In this case, $k' = k - |Z|$. We claim that $w(x') \geq w(x) - |Z|$. Suppose that this is not true. Then, it must be the case that $w(x') \leq w(x) - |Z| - \frac{1}{2}$. We will now consider three cases depending on the value $x'(z)$ where z is the vertex in G' resulting from the identification of $N(Z)$.

   Case 1: $x'(z) = 1$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$. For every vertex in $N(Z)$, assign 1 and for every vertex in Z, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') - 1 + |N(Z)| = w(x') - 1 + (|Z| + 1) \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

   Case 2: $x'(z) = 0$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$. For every vertex in Z, assign 1 and for every vertex in $N(Z)$, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') + |Z| \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

   Case 3: $x'(z) = \frac{1}{2}$. Now consider the following function $x'' : V \rightarrow \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$.

For every vertex in $Z \cup N(Z)$, assign $\frac{1}{2}$. Clearly this is a feasible solution. But now, $w(x'') = w(x') - \frac{1}{2} + \frac{1}{2}(|Z| + |N(Z)|) = w(x') - \frac{1}{2} + \frac{1}{2}(|Z| + |Z| + 1) \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

Hence, $w(x') \geq w(x) - |Z|$, which implies that $\mu(G', k') \leq \mu(G, k)$.

4. We now consider the branching step.

   (a) Consider the case when we pick u in the vertex cover. In this case, $k' = k - 1$. We claim that $w(x') \geq w(x) - \frac{1}{2}$. Suppose that this is not the case. Then, it must be the case that $w(x') \leq w(x) - 1$. Consider the following function $x'' : V \to \{0, \frac{1}{2}, 1\}$. For every vertex $v \in V \setminus \{u\}$, $x''(v) = x'(v)$ and $x''(u) = 1$. Now, x'' is clearly a feasible solution for LPVC(G) and has a value at most that of x. But this contradicts our assumption that $x \equiv \frac{1}{2}$ is the unique optimum solution to LPVC(G). Hence, $w(x') \geq w(x) - \frac{1}{2}$, which implies that $\mu(G', k') \leq \mu(G, k) - \frac{1}{2}$.

   (b) Consider the case when we don't pick u in the vertex cover. In this case, $k' = k - |N(u)|$. We know that $w(x') = w(x) - \frac{1}{2}(|\{u\}| + |N(u)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(u)|)$, we get $w(x') = w(x) - |N(u)| - \frac{1}{2}(|\{u\}| - |N(u)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. But, $\{u\} \cup V_0^{x'}$ is an independent set in G, and $N(\{u\} \cup V_0^{x'}) = N(u) \cup V_1^{x'}$ in G. Since $\mathbf{surplus}(G) \geq 2$, $|N(\{u\} \cup V_0^{x'})| - |\{u\} \cup V_0^{x'}| \geq 2$. Hence, $w(x') = w(x) - |N(u)| + \frac{1}{2}(|N(\{u\} \cup V_0^{x'})| - |\{u\} \cup V_0^{x'}|) \geq w(x) - |N(u)| + 1$.

   Hence, $\mu(G', k') \leq \mu(G, k) - 1$.

We have thus shown that the preprocessing rules do not increase the measure $\mu = \mu(G, k)$ and the branching step results in a $(\frac{1}{2}, 1)$ branching vector, resulting in the recurrence $T(\mu) \leq T(\mu - \frac{1}{2}) + T(\mu - 1)$ which solves to $(2.6181)^{\mu} = (2.6181)^{k - vc^*(G)}$. Thus, we get a $\mathcal{O}^*(2.6181^{(k-vc^*(G))})$ algorithm for Vertex Cover above LP.

Theorem 6.3.8. Vertex Cover above LP *can be solved in time* $\mathcal{O}^*((2.6181)^{k-vc^*(G)})$.

By applying the above theorem iteratively for increasing values of k, we can compute a minimum vertex cover of G and hence we have the following corollary.

Corollary 6.3.9. *There is an algorithm that, given a graph G, runs in time* $\mathcal{O}^*(2.6181^{(vc(G)-vc^*(G))})$ *and computes a minimum vertex cover of G.*

6.4 Improved Algorithm for Vertex Cover above LP

In this section we give an improved algorithm for Vertex Cover above LP using some more branching steps based on the structure of the neighborhood of the vertex (set) on which we branch. The goal is to achieve branching vectors better that $(\frac{1}{2}, 1)$.

6.4.1 Some general claims to measure the drops

First, we capture the drop in the measure in the branching steps, including when we branch on a larger sized sets. In particular, when we branch on a set S of vertices, in one branch we set all vertices of S to 1, and in the other, we set all vertices of S to 0. Note, however that such a branching on S may not be exhaustive (as the branching doesn't explore the possibility that some vertices of S are set to 0 and some are set to 1) unless the set S satisfies the premise of Lemma 6.3.3. Let $\mu = \mu(G, k)$ be the measure as defined in the previous section.

Lemma 6.4.1. *Let G be a graph with* $\textbf{surplus}(G) = p$, *and let S be an independent set. Let $\mathcal{H}_S$ be the collection of all independent sets of G that contain S (including S). Then,* *including S in the vertex cover while branching leads to a decrease of* $\min\{\frac{|S|}{2}, \frac{p}{2}\}$ *in μ; and the branching excluding S from the vertex cover leads to a drop of* $\frac{\mathbf{surplus}(\mathcal{H}_S)}{2} \geq \frac{p}{2}$ *in μ.*

Proof. Let (G', k') be the instance resulting from the branching, and let x' be an optimum solution to LPVC(G'). Consider the case when we pick S in the vertex cover. In this case, $k' = k - |S|$. We know that $w(x') = w(x) - \frac{|S|}{2} + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. If $V_0^{x'} = \emptyset$, then we know that $V_1^{x'} = \emptyset$, and hence we have that $w(x') = w(x) - \frac{|S|}{2}$. Else, by adding and subtracting $\frac{1}{2}(|S|)$, we get $w(x') = w(x) - |S| + \frac{|S|}{2} + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. However, $N(V_0^{x'}) \subseteq S \cup V_1^{x'}$ in G. Thus, $w(x') \geq w(x) - |S| + \frac{1}{2}(|N(V_0^{x'})| - |V_0^{x'}|)$. We also know that $V_0^{x'}$ is an independent set in G, and thus, $|N(V_0^{x'})| - |V_0^{x'}| \geq \mathbf{surplus}(G) = p$. Hence, in the first case $\mu(G', k') \leq \mu(G, k) - \frac{|S|}{2}$ and in the second case $\mu(G', k') \leq \mu(G, k) - \frac{p}{2}$. Thus, the drop in the measure when S is included in the vertex cover is at least $\min\{\frac{|S|}{2}, \frac{p}{2}\}$.

Consider the case when we don't pick S in the vertex cover. In this case, $k' = k - |N(S)|$. We know that $w(x') = w(x) - \frac{1}{2}(|S| + |N(S)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(S)|)$, we get $w(x') = w(x) - |N(S)| + \frac{1}{2}(|N(S)| - |S|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. But, $S \cup V_0^{x'}$ is an independent set in G, and $N(S \cup V_0^{x'}) = N(S) \cup V_1^{x'}$ in G. Thus, $|N(S \cup V_0^{x'})| - |S \cup V_0^{x'}| \geq \mathbf{surplus}(\mathcal{H}_S)$. Hence, $w(x') = w(x) - |N(S)| + \frac{1}{2}(|N(S \cup V_0^{x'})| - |S \cup V_0^{x'}|) \geq w(x) - |N(S)| + \frac{\mathbf{surplus}(\mathcal{H}_S)}{2}$. Hence, $\mu(G', k') \leq \mu(G, k) - \frac{\mathbf{surplus}(\mathcal{H}_S)}{2}$. □

Thus, after the preprocessing steps (when the surplus of the graph is at least 2), suppose we manage to find (in polynomial time) a set $S \subseteq V$ such that

- **surplus**(G) = **surplus**(S) = **surplus**$(\mathcal{H}_S)$,
- $|S| \geq 2$, and

- that the branching that sets all of S to 0 or all of S to 1 is exhaustive.

Then, Lemma 6.4.1 guarantees that branching on this set right away leads to a $(1, 1)$ branching vector. We now explore the cases in which such sets do exist. Note that the first condition above implies the third from the Lemma 6.3.3. First, we show that if there exists a set S such that $|S| \geq 2$ and **surplus** (G) = **surplus**(S), then we can find such a set in polynomial time.

Lemma 6.4.2. *Let G be a graph on which Preprocessing Rule 1 does not apply (i.e. all $\frac{1}{2}$ is the unique solution to LPVC(G)). If G has an independent set S' such that $|S'| \geq 2$ and* $\textbf{surplus}(S') = \textbf{surplus}(G)$*, then in time $\mathcal{O}(mn^2\sqrt{n})$ we can find an independent set S such that $|S| \geq 2$ and* $\textbf{surplus}(S) = \textbf{surplus}(G)$.

Proof. By our assumption we know that G has an independent set S' such that $|S'| \geq 2$ and $\textbf{surplus}(S') = \textbf{surplus}(G)$. Let $u, v \in S'$. Let $\mathcal{H}$ be the collection of all independent sets of G containing u and v. Let x be an optimal solution to LPVC(G) obtained after setting $x(u) = 0$ and $x(v) = 0$. Take $S = V_0^x$, clearly, we have that $\{u, v\} \subseteq V_0^x$. We now have the following claim.

Claim 3. $\textbf{surplus}(S)$ = $\textbf{surplus}(G)$.

Proof. We know that the objective value of LPVC(G) after setting $x(u) = x(v) = 0$, $w(x) = |V|/2 + (|N(S)| - |S|)/2 = |V|/2 + \textbf{surplus}(S)/2$, as all $\frac{1}{2}$ is the unique solution to LPVC(G).

Another solution x', for LPVC(G) that sets u and v to 0, is obtained by setting $x'(a) = 0$ for every $a \in S'$, $x'(a) = 1$ for every $a \in N(S')$ and by setting all other variables to $1/2$. It is easy to see that such a solution is a feasible solution of the required kind and $w(x') = |V|/2 + (|N(S')| - |S'|)/2 = |V|/2 + \textbf{surplus}(S')/2$. However, as x is also an optimum solution, $w(x) = w(x')$, and hence we have that $\textbf{surplus}(\textbf{S}) \leq$

$\mathbf{surplus}(\mathbf{S}')$. But as S' is a set of minimum surplus in G, we have that $\mathbf{surplus}(S) = \mathbf{surplus}(S') = \mathbf{surplus}(G)$ proving the claim. $\square$

Thus, we can find a such a set S in polynomial time by solving LPVC(G) after setting $x(u) = 0$ and $x(v) = 0$ for every pair of vertices u, v such that $(u, v) \notin E$ and picking that set V_0^x which has the minimum surplus among all $x's$ among all pairs u, v. Since any V_0^x contains at least 2 vertices, we have that $|S| \geq 2$. The bound on the time required to find this set follows from Lemma 6.3.1. $\square$

6.4.2 (1,1) drops in the measure

Lemma 6.4.1 and Lemma 6.4.2 together imply that, if there is a minimum surplus set of size at least 2 in the graph, then we can find and branch on that set to get a $(1, 1)$ drop in the measure.

Suppose that there is no minimum surplus set of size more than 1. Note that, by Lemma 6.4.1, when $\mathbf{surplus}(G) \geq 2$, we get a drop of $(\mathbf{surplus}(\mathbf{G}))/2 \geq 1$ in the branch where we *exclude* a vertex or a set. Hence, if we find a vertex (set) to exclude in either branch of a two way branching, we get a $(1, 1)$ branching vector. We now identify another such case.

Lemma 6.4.3. *Let v be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique for some neighbor u of v. Then, there exists a minimum vertex cover that excludes either v or u.*

Proof. Towards the proof we first show the following well known observation.

Claim 4. *Let G be a graph and v be a vertex. Then there exists a minimum vertex cover for G containing $N(v)$ or at most $|N(v)| - 2$ vertices from $N(v)$.*

Proof. If a minimum vertex cover of G, say C, contains exactly $|N(v)| - 1$ vertices of $N(v)$, then we know that C must contain v. Observe that $C' = C \setminus \{v\} \cup N(v)$

is also a vertex cover of G of the same size as C. However, in this case, we have a minimum vertex cover containing $N(v)$. Thus, there exists a minimum vertex cover of G containing $N(v)$ or at most $|N(v)| - 2$ vertices from $N(v)$. $\square$

Let v be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique. Consider a minimum vertex cover and suppose that v is in the vertex cover. Clearly, $N(v)$ is not contained in this vertex cover. Since $G[N(v) \setminus \{u\}]$ is a clique this vertex cover contains at least $|N(v)| - 2$ vertices from $G[N(v) \setminus \{u\}]$. Hence, by Claim 4, the vertex u is not part of the vertex cover. This completes the proof. $\square$

Next, in order to identify another case where we might obtain a $(1, 1)$ branching vector, we first observe and capture the fact that when Preprocessing Rule 2 is applied, the measure $k - vc^*(G)$ actually drops by at least $\frac{1}{2}$ (as proved in item 2 of the analysis of the simple algorithm in Section 6.3.3).

Lemma 6.4.4. *Let (G, k) be the input instance and (G', k') be the instance obtained after applying Preprocessing Rule 2. Then, $\mu(G', k') \leq \mu(G, k) - \frac{1}{2}$.*

Thus, after we branch on an arbitrary vertex, if we are able to apply Preprocessing Rule 2 in the branch where we include that vertex, we get a $(1, 1)$ drop. This is because, in the branch where we exclude the vertex, we get a drop of 1 by Lemma 6.4.1, and in the branch where we include the vertex, we get a drop of $\frac{1}{2}$ by Lemma 6.4.1, which is then followed by a drop of $\frac{1}{2}$ due to Lemma 6.4.4.

Thus, after preprocessing, the algorithm performs the following steps (see Figure 6.2) each of which results in a $(1, 1)$ drop as argued before. Note that Preprocessing Rule 1 cannot apply in the graph $G \setminus \{v\}$ since the surplus of G can drop by at most 1 by deleting a vertex. Hence, checking if rule **B**3 applies is equivalent to checking if, for some vertex v, Preprocessing Rule 2 applies in the graph $G \setminus \{v\}$. Recall that, by Lemma 6.3.5

we can check this in polynomial time and hence we can check if **B**3 applies on the graph in polynomial time.

Branching Rules.
These branching rules are applied in this order.

B 1. Apply Lemma 6.4.2 to test if there is a set S such that **surplus**(S)=**surplus**(G) and $|S| \geq 2$. If so, then branch on S.

B 2. Let v be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique for some vertex u in $N(v)$. Then in one branch add $N(v)$ into the vertex cover, decrease k by $|N(v)|$, and delete $N[v]$ from the graph. In the other branch add $N(u)$ into the vertex cover, decrease k by $|N(u)|$, and delete $N[u]$ from the graph.

B 3. Apply Lemma 6.3.5 to test if there is a vertex v such that Preprocessing Rule 2 applies in $G \setminus \{v\}$. If there is such a vertex, then branch on v.

Figure 6.2: Outline of the branching steps yielding $(1, 1)$ drop.

6.4.3 A Branching step yielding $(1/2, 3/2)$ drop

Now, suppose none of the preprocessing and branching rules presented thus far apply. Let v be a vertex with degree at least 4. Let $S = \{v\}$ and recall that $\mathcal{H}_S$ is the collection of all independent sets containing S, and **surplus** $(\mathcal{H}_S)$ is the surplus of an independent set with minimum surplus in $\mathcal{H}_S$. We claim that $\textbf{surplus}(\mathcal{H}_S) \geq 3$.

As the preprocessing rules don't apply, clearly $\textbf{surplus}(\mathcal{H}_S) \geq \textbf{surplus}(G) \geq 2$. If $\textbf{surplus}(\mathcal{H}_S) = 2$, then the set that realizes $\textbf{surplus}(\mathcal{H}_S)$ is not S (as the $\textbf{surplus}(S) = degree(v) - 1 = 3$), but a superset of S, which is of cardinality at least 2. Then, the Branching Rule **B**1 would have applied which is a contradiction. This proves the claim. Hence, by Lemma 6.4.1, we get a drop of at least $3/2$ in the branch that excludes the vertex v resulting in a $(1/2, 3/2)$ drop. This branching step is illustrated in Figure 6.3.

B 4. If there exists a vertex v of degree at least 4 then branch on v.

Figure 6.3: The branching step yielding a $(1/2, 3/2)$ drop.

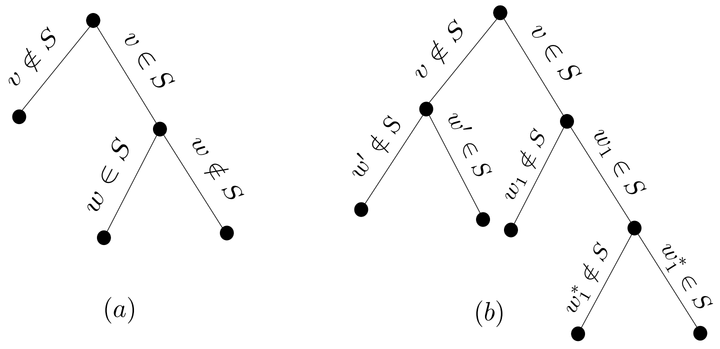

Figure 6.4: Illustrations of the branches of rules (a) **B**5 and (b) **B**6

6.4.4 A Branching step yielding $(1, 3/2, 3/2)$ drop

Next, we observe that when branching on a vertex, if in the branch that includes the vertex in the vertex cover (which guarantees a drop of $1/2$), any of the Branching Rules **B**1 or **B**2 or **B**3 applies, then combining the subsequent branching with this branch of the current branching step results in a net drop of $(1, 3/2, 3/2)$ (which is $(1, 1/2+1, 1/2+1)$) (see Figure 6.10 (a)). Thus, we add the following branching rule to the algorithm (Figure 6.5).

B 5. Let v be a vertex. If **B**1 applies in $\mathcal{R}(G \setminus \{v\})$ or there exists a vertex w in $\mathcal{R}(G \setminus \{v\})$ on which either **B**2 or **B**3 applies then branch on v.

Figure 6.5: The branching step yielding a $(1, 3/2, 3/2)$ drop.

6.4.5 The Final branching step

Finally, if the preprocessing and branching rules presented thus far do not apply, then note that we are left with a 3-regular graph. In this case, we simply pick a vertex v and branch. However, we execute the branching step carefully in order to simplify the analysis of the drop. More precisely, we execute the following step at the end.

B 6. Pick an arbitrary degree 3 vertex v in G and let x, y and z be the neighbors of v. Then in one branch add v into the vertex cover, decrease k by 1, and delete v from the graph. The other branch that excludes v from the vertex cover, is performed as follows. Delete x from the graph, decrease k by 1, and obtain $\mathcal{R}(G \setminus \{x\})$. During the process of obtaining $\mathcal{R}(G \setminus \{x\})$, Preprocessing Rule 3 would have been applied on vertices y and z to obtain a 'merged' vertex v_{yz} (see proof of correctness of this rule). Now delete v_{yz} from the graph $\mathcal{R}(G \setminus \{x\})$, and decrease k by 1.

Figure 6.6: Outline of the last step.

6.4.6 Complete Algorithm and Correctness

A detailed outline of the algorithm is given in Figure 6.7. Note that we have already argued the correctness and analyzed the drops of all steps except the last step, **B**6.
The correctness of this branching rule will follow from the fact that $\mathcal{R}(G \setminus \{x\})$ is obtained by applying Preprocessing Rule 3 alone and that too only on the neighbors of x, that is, on the degree 2 vertices of $G \setminus \{x\}$ (Lemma 6.4.10). Lemma 6.4.15 (to appear later) shows the correctness of deleting v_{yz} from the graph $\mathcal{R}(G \setminus \{x\})$ without branching. Thus, the correctness of this algorithm follows from the soundness of the preprocessing rules and the fact that the branching is exhaustive.

The running time will be dominated by the way **B**6 and the subsequent branching

Preprocessing Step. Apply Preprocessing Rules 1, 2 and 3 in this order exhaustively on G.

Connected Components. Apply the algorithm on connected components of G separately. Furthermore, if a connected component has size at most 10, then solve the problem optimally in $\mathcal{O}(1)$ time.

Branching Rules.
These branching rules are applied in this order.

B1 If there is a set S such that **surplus**(S)=**surplus**(G) and $|S| \geq 2$, then branch on S.

B2 Let v be a vertex such that $G[N(v) \setminus \{u\}]$ is a clique for some vertex u in $N(v)$. Then in one branch add $N(v)$ into the vertex cover, decrease k by $|N(v)|$, and delete $N[v]$ from the graph. In the other branch add $N(u)$ into the vertex cover, decrease k by $|N(u)|$, and delete $N[u]$ from the graph.

B3 Let v be a vertex. If Preprocessing Rule 2 can be applied to obtain $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$, then branch on v.

B4 If there exists a vertex v of degree at least 4 then branch on v.

B5 Let v be a vertex. If **B**1 applies in $\mathcal{R}(G \setminus \{v\})$ or if there exists a vertex w in $\mathcal{R}(G \setminus \{v\})$ on which **B**2 or **B**3 applies then branch on v.

B6 Pick an arbitrary degree 3 vertex v in G and let x, y and z be the neighbors of v. Then in one branch add v into the vertex cover, decrease k by 1, and delete v from the graph. The other branch, the one that excludes v from the vertex cover, is performed as follows. Delete x from the graph, decrease k by 1, and obtain $\mathcal{R}(G \setminus \{x\})$. Now, delete v_{yz} from the graph $\mathcal{R}(G \setminus \{x\})$, the vertex that has been created by the application of Preprocessing Rule 3 on v while obtaining $\mathcal{R}(G \setminus \{x\})$ and decrease k by 1.

Figure 6.7: Outline of the Complete algorithm.

apply. We will see that **B**6 is our most expensive branching rule. In fact, this step dominates the running time of the algorithm of $\mathcal{O}^*(2.3146^{\mu(G,k)})$ due to a branching vector of $(3/2, 3/2, 5/2, 5/2, 2)$. We will argue that when we apply **B**6 on a vertex, say v, then on either side of the branch we will be able to branch using rules **B**1, or **B**2, or

Rule	**B**1	**B**2	**B**3	**B**4	**B**5	**B**6
Branching Vector	(1,1)	(1,1)	(1,1)	$(\frac{1}{2}, \frac{3}{2})$	$(\frac{3}{2}, \frac{3}{2}, 1)$	$(\frac{3}{2}, \frac{3}{2}, \frac{5}{2}, \frac{5}{2}, 2)$
Running time	2^μ	2^μ	2^μ	2.1479^μ	2.3146^μ	2.3146^μ

Figure 6.8: A table giving the decrease in the measure due to each branching rule.

B3 or **B**4. More precisely, we show that in the branch where we include v in the vertex cover,

- there is a vertex of degree 4 in $\mathcal{R}(G \setminus \{v\})$. Thus, **B**4 will apply on the graph $\mathcal{R}(G \setminus \{v\})$ (if any of the earlier branching rules applied in this graph, then rule **B**5 would have applied on G).
- $\mathcal{R}(G \setminus \{v\})$ has a degree 4 vertex w such that there is a vertex of degree 4 in the graph $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w\})$ and thus one of the Branching Rules **B**1, **B**2, **B**3 or **B**4 applies on the graph $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w\})$.

Similarly, in the branch where we exclude the vertex v from the solution (and add the vertices x and v_{yz} into the vertex cover), we will show that a degree 4 vertex remains in the reduced graph. This yields the claimed branching vector (see Figure 6.4.6). The rest of the section is geared towards showing this.

We start with the following definition.

Definition 6.4.5. *We say that a graph G is* irreducible *if Preprocessing Rules 1, 2 and 3 and the Branching Rules* **B***1,* **B***2,* **B***3,* **B***4 and* **B***5 do not apply on G.*

Observe that when we apply **B**6, the current graph is 3-regular. Thus, after we delete a vertex v from the graph G and apply Preprocessing Rule 3 we will get a degree 4 vertex. Our goal is to identify conditions that ensure that the degree 4 vertices we obtain by applying Preprocessing Rule 3 survive in the graph $\mathcal{R}(G \setminus \{v\})$. We prove the existence of degree 4 vertices in subsequent branches after applying **B**6 as follows.

- We do a closer study of the way Preprocessing Rules 1, 2 and 3 apply on $G \setminus \{v\}$ if Preprocessing Rules 1, 2 and 3 and the Branching Rules **B**1, **B**2 and **B**3 do not apply on G. Based on our observations, we prove some structural properties of the graph $\mathcal{R}(G \setminus \{v\})$, This is achieved by Lemma 6.4.10.

- Next, we show that Lemma 6.4.10, along with the fact that the graph is irreducible implies a lower bound of 7 on the length of the shortest cycle in the graph (Lemma 6.4.13). This lemma allows us to argue that when the preprocessing rules are applied, their effect is local.

- Finally, Lemmas 6.4.10 and 6.4.13 together ensure the presence of the required number of degree 4 vertices in the subsequent branching (Lemma 6.4.14).

Main Structural Lemmas: Lemmas 6.4.10 and 6.4.13

We start with some simple well known observations that we use repeatedly in this section. These observations follow from results in [84]. We give proofs for completeness.

Lemma 6.4.6. *Let G be an undirected graph, then the following are equivalent.*

(1) Preprocessing Rule 1 applies (i.e. All $\frac{1}{2}$ is not the unique solution to the LPVC(G).)

(2) There exists an independent set I of G such that $\mathbf{surplus}(I) \leq 0$.

(3) There exists an optimal solution x to LPVC(G) that assigns 0 to some vertex.

Proof. $(1) \implies (3)$: As we know that the optimum solution is half-integral, there exists an optimum solution that assigns 0 or 1 to some vertex. Suppose no vertex is assigned 0. Then, for any vertex which is assigned 1, its value can be reduced to $\frac{1}{2}$ maintaining feasibility (as all its neighbors have been assigned value $\geq \frac{1}{2}$) which is a contradiction to the optimality of the given solution.

$(3) \implies (2)$: Let $I = V_0^x$, and suppose that $\mathbf{surplus}(I) > 0$. Then consider the solution x' that assigns $1/2$ to vertices in $I \cup N(I)$, retaining the value of x for the other vertices. Then x' is a feasible solution whose objective value $w(x')$ drops from $w(x)$ by $(|N(I)| - |I|)/2 = \mathbf{surplus}(I)/2 > 0$ which is a contradiction to the optimality of x.

$(2) \implies (1)$: Setting all vertices in I to 0, all vertices in $N(I)$ to 1 and setting the remaining vertices to $\frac{1}{2}$ gives a feasible solution whose objective value is at most $|V|/2$, and hence all $\frac{1}{2}$ is not the unique solution to LPVC(G). □

Lemma 6.4.7. *Let G be an undirected graph, then the following are equivalent.*

(1) Preprocessing Rule 1 or 2 or 3 applies.

(2) There exists an independent set I such that $\mathbf{surplus}(I) \leq 1$.

(3) There exists a vertex v such that an optimal solution x to LPVC($G \setminus \{v\}$) assigns 0 *to some vertex.*

Proof. The fact that (1) and (2) are equivalent follows from the definition of the preprocessing rules and Lemma 6.4.6.

$(3) \implies (2)$. By Lemma 6.4.6, there exists an independent set I in $G \setminus \{v\}$ whose surplus is at most 0. The same set will have surplus at most 1 in G.

$(2) \implies (3)$. Let $v \in N(I)$. Then I is an independent set in $G \setminus \{v\}$ with surplus at most 0, and hence by Lemma 6.4.6, there exists an optimal solution to LPVC($G \setminus \{v\}$) that assigns 0 to some vertex. □

We now prove an auxiliary lemma about the application of Preprocessing Rule 3 which will be useful in simplifying later proofs.

Lemma 6.4.8. *Let G be a graph and G_R be the graph obtained from G by applying Preprocessing Rule 3 on an independent set Z. Let z denote the newly added vertex corresponding to Z in G_R.*

1. *If G_R has an independent set I such that* $\mathbf{surplus}(I) = p$*, then G also has an independent set I' such that* $\mathbf{surplus}(I') = p$ *and $|I'| \geq |I|$.*

2. *Furthermore, if $z \in I \cup N(I)$ then $|I'| > |I|$.*

Proof. Let Z denote the minimum surplus independent set on which Preprocessing Rule 3 has been applied and z denote the newly added vertex. Observe that since Preprocessing Rule 3 applies on Z, we have that Z and $N(Z)$ are independent sets, $|N(Z)| = |Z| + 1$ and $|N(Z)| \geq 2$.

Let I be an independent set of G_R such that $\mathbf{surplus}(I) = p$.

- If both I and $N(I)$ do not contain z then we have that G has an independent set I such that $\mathbf{surplus}(I) = p$.

- Suppose $z \in I$. Then consider the following set: $I' := I \setminus \{z\} \cup N(Z)$. Notice that z represents $N(Z)$ and thus I do not have any neighbors of $N(Z)$. This implies that I' is an independent set in G. Now we will show that $\mathbf{surplus}(I') = p$. We know that $|N(Z)| = |Z| + 1$ and $N(I') = N(I) \cup Z$. Thus,

$$\begin{aligned}
|N(I')| - |I'| &= (|N(I)| + |Z|) - |I'| \\
&= (|N(I)| + |Z|) - (|I| - 1 + |N(Z)|) \\
&= (|N(I)| + |Z|) - (|I| + |Z|) \\
&= |N(I)| - |I| = \mathbf{surplus}(I) = p.
\end{aligned}$$

- Suppose $z \in N(I)$. Then consider the following set: $I' := I \cup Z$. Notice that z represents $N(Z)$ and since $z \notin I$ we have that I do not have any neighbors of Z. This implies that I' is an independent set in G. We show that $\textbf{surplus}(I') = p$. We know that $|N(Z)| = |Z| + 1$. Thus,

$$\begin{aligned} |N(I')| - |I'| &= (|N(I)| - 1 + |N(Z)|) - |I'| \\ &= (|N(I)| - 1 + |N(Z)|) - (|I| + |Z|) \\ &= (|N(I)| + |Z|) - (|I| + |Z|) \\ &= |N(I)| - |I| = \textbf{surplus}(I) = p. \end{aligned}$$

From the construction of I', it is clear that $|I'| \geq |I|$ and if $z \in (I \cup N(I))$ then $|I'| > |I|$. This completes the proof. □

We now give some definitions that will be useful in formulating the statement of the main structural lemma.

Definition 6.4.9. *Let G be a graph and $\mathcal{P} = P_1, P_2, \cdots, P_\ell$ be a sequence of exhaustive applications of Preprocessing Rules 1, 2 and 3 applied in this order on G to obtain G'. Let $\mathcal{P}_3 = P_a, P_b, \cdots, P_t$ be the subsequence of $\mathcal{P}$ restricted to Preprocessing Rule 3. Furthermore let Z_j, $j \in \{a, \ldots, t\}$ denote the minimum surplus independent set corresponding to P_t on which the Preprocessing Rule 3 has been applied and z_j denote the newly added vertex (See Lemma 6.3.4). Let $Z^* = \{z_j \mid j \in \{a, \ldots, t\}\}$ be the set of these newly added vertices.*

- *We say that an application of Preprocessing Rule 3 is trivial if the minimum surplus independent set Z_j on which P_j is applied has size 1, that is, $|Z_j| = 1$.*

- *We say that all applications of Preprocessing Rule 3 are independent if for all $j \in \{a, \ldots, t\}$, $N[Z_j] \cap Z^* = \emptyset$.*

Essentially, independent applications of Preprocessing Rule 3 mean that the set on which the rule is applied, as well as all its neighbors are vertices in the original graph.

Next, we state and prove one of the main structural lemmas of this section.

Lemma 6.4.10. *Let $G = (V, E)$ be a graph on which Preprocessing Rules 1, 2 and 3 and the Branching Rules* **B**1*,* **B**2 *and* **B**3 *do not apply. Then for any vertex $v \in V$,*

1. *Preprocessing Rules 1 and 2 have not been applied while obtaining $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$;*
2. *and all applications of the Preprocessing Rule 3 while obtaining $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$ are independent and trivial.*

Proof. Fix a vertex v. Let $G_0 = G \setminus \{v\}, G_1, \ldots, G_t = \mathcal{R}(G \setminus \{v\})$ be a sequence of graphs obtained by applying Preprocessing Rules 1, 2 and 3 in this order to obtain the reduced graph $\mathcal{R}(G \setminus \{v\})$.

We first observe that Preprocessing Rule 2 never applies in obtaining $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$ since otherwise, **B**3 would have applied on G. Next, we show that Preprocessing Rule 1 does not apply. Let q be the least integer such that Preprocessing Rule 1 applies on G_q and it does not apply to any graph $G_{q'}$, $q' < q$. Suppose that $q \geq 1$. Then, only Preprocessing Rule 3 has been applied on $G_0, \ldots, G_{q-1}$. This implies that G_q has an independent set I_q such that $\mathbf{surplus}(I_q) \leq 0$. Then, by Lemma 6.4.8, G_{q-1} also has an independent set I'_q such that $\mathbf{surplus}(I'_q) \leq 0$ and thus by Lemma 6.4.6 Preprocessing Rule 1 applies to G_{q-1}. This contradicts the assumption that on G_{q-1} Preprocessing Rule 1 does not apply. Thus, we conclude that q must be zero. So, $G \setminus \{v\}$ has an independent set I_0 such that $\mathbf{surplus}(I_0) \leq 0$ in $G \setminus \{v\}$ and thus I_0 is an independent set in G such that $\textbf{surplus}(I_0) \leq 1$ in G. By Lemma 6.4.7 this implies that either of Preprocessing Rules 1, 2 or 3 is applicable on G, a contradiction to the given assumption.

Now we show the second part of the lemma. By the first part we know that the G_i's have been obtained by applications of Preprocessing Rule 3 alone. Let Z_i, $0 \leq i \leq t-1$ be the sets in G_i on which Preprocessing Rule 3 has been applied. Let the newly added vertex corresponding to $N(Z_i)$ in this process be z_i'. We now make the following claim.

Claim 5. *For any $i \geq 0$, if G_i has an independent set I_i such that* $\textbf{surplus}(I_i) = 1$, *then G has an independent set I such that $|I| \geq |I_i|$ and* $\textbf{surplus}(I) = 2$. *Furthermore, if $(I_i \cup N(I_i)) \cap \{z_1, \ldots, z_{i-1}\} \neq \phi$, then $|I| > |I_i|$.*

Proof. We prove the claim by induction on the length of the sequence of graphs. For the base case consider $q = 0$. Since Preprocessing Rules 1, 2, and 3 do not apply on G, we have that $\textbf{surplus}(G) \geq 2$. Since I_0 is an independent set in $G \setminus \{v\}$ we have that I_0 is an independent set in G also. Furthermore since $\textbf{surplus}(I_0) = 1$ in $G \setminus \{v\}$, we have that $\textbf{surplus}(I_0) = 2$ in G, as $\textbf{surplus}(G) \geq 2$. This implies that G has an independent set I_0 with $\textbf{surplus}(I_0) = 2 = \textbf{surplus}(G)$. Furthermore, since G_0 does not have any newly introduced vertices, the last assertion is vacuously true. Now let $q \geq 1$. Suppose that G_q has a set $|I_q|$ and $\textbf{surplus}(I_q) = 1$. Thus, by Lemma 6.4.8, G_{q-1} also has an independent set I_q' such that $|I_q'| \geq |I_q|$ and $\textbf{surplus}(I_q') = 1$. Now by the induction hypothesis, G has an independent set I such that $|I| \geq |I_q'| \geq |I_q|$ and $\textbf{surplus}(I) = 2 = \textbf{surplus}(G)$.

Next we consider the case when $(I_q \cup N(I_q)) \cap \{z_1', \ldots, z_{q-1}'\} \neq \emptyset$. If $z_{q-1}' \notin I_q \cup N(I_q)$ then we have that I_q is an independent set in G_{q-1} such that $(I_q \cup N(I_q)) \cap \{z_1', \ldots, z_{q-2}'\} \neq \emptyset$. Thus, by the induction hypothesis, we have that G has an independent set I such that $|I| > |I_q|$ and $\textbf{surplus}(I) = 2 = \textbf{surplus}(G)$. On the other hand, if $z_{q-1}' \in I_q \cup N(I_q)$ then by Lemma 6.4.8, we know that G_{q-1} has an independent set I'_q such that $|I'_q| > |I_q|$ and $\mathbf{surplus}(I'_q) = 1$. Now by induction hypothesis we know that G has an independent set I such that $|I| \geq |I'_q| > |I_q|$ and $\mathbf{surplus}(I) = 2 = \mathbf{surplus}(G)$. This concludes the proof of the claim. $\square$

We first show that all the applications of Preprocessing Rule 3 are trivial. Claim 5 implies that if we have a non-trivial application of Preprocessing Rule 3 then it implies that G has an independent set I such that $|I| \geq 2$ and $\mathbf{surplus}(I) = 2 = \mathbf{surplus}(G)$. Then, **B**1 would apply on G, a contradiction.

Finally, we show that all the applications of Preprocessing Rule 3 are independent. Let q be the least integer such that the application of Preprocessing Rule 3 on G_q is not independent. That is, the application of Preprocessing Rule 3 on $G_{q'}$, $q' < q$, is trivial and independent. Observe that $q \geq 1$. We already know that every application of Preprocessing Rule 3 is trivial. This implies that the set Z_q contains a single vertex. Let $Z_q = \{z_q\}$. Since the application of Preprocessing Rule 3 on Z_q is not independent we have that $(Z_q \cup N(Z_q)) \cap \{z'_1, \cdots, z'_{q-1}\} \neq \emptyset$. We also know that $\mathbf{surplus}(Z_q) = 1$ and thus by Claim 5 we have that G has an independent set I such that $|I| \geq 2 > |Z_q|$ and $\mathbf{surplus}(I) = 2 = \mathbf{surplus}(G)$. This implies that **B**1 would apply on G, a contradiction. Hence, we conclude that all the applications of Preprocessing Rule 3 are independent. This proves the lemma. $\square$

Let $g(G)$ denote the girth of the graph, that is, the length of the smallest cycle in G. Our next goal of this section is to obtain a lower bound on the girth of an irreducible graph. Towards this, we first introduce the notion of an *untouched* vertex.

Definition 6.4.11. *We say that a vertex v is* untouched *by an application of Preprocessing Rule 2 or Preprocessing Rule 3, if $\{v\} \cap (Z \cup N(Z)) = \emptyset$, where Z is the set on which the rule is applied.*

We now prove an auxiliary lemma regarding the application of the preprocessing rules on graphs of a certain girth and following that, we will prove a lower bound on the girth of irreducible graphs.

Lemma 6.4.12. *Let G be a graph on which Preprocessing Rules 1, 2 and 3 and the Branching Rules* **B**1*,* **B**2*,* **B**3 *do not apply and suppose that $g(G) \geq 5$. Then for any vertex $v \in V$, any vertex $x \notin N_2[v]$ is untouched by the preprocessing rules applied to obtain the graph $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$ and has the same degree as it does in G.*

Proof. Since the preprocessing rules do not apply in G, the minimum degree of G is at least 3 and since the graph G does not have cycles of length 3 or 4, for any vertex v, the neighbors of v are independent and there are no edges between vertices in the first and second neighborhood of v.

We know by Lemma 6.4.10 that only Preprocessing Rule 3 applies on the graph $G \setminus \{v\}$ and it applies only in a trivial and independent way. Let $U = \{u_1, \ldots, u_t\}$ be the degree 3 neighbors of v in G and let D represent the set of the remaining (high degree) neighbors of v. Let $P_1, \ldots, P_l$ be the sequence of applications of Preprocessing Rule 3 on the graph $G \setminus \{v\}$, let Z_i be the minimum surplus set corresponding to the application of P_i and let z_i be the new vertex created during the application of P_i.

We prove by induction on i, that

- the application P_i corresponds to a vertex $u_j \in U$,
- any vertex $x \notin N_2[v] \setminus D$ is untouched by this application, and
- after the application of P_i, the degree of $x \notin N_2[v]$ in the resulting graph is the same as that in G.

In the base case, $i = 1$. Clearly, the only vertices of degree 2 in the graph $G \setminus \{v\}$ are the degree 3 neighbors of v. Hence, the application P_1 corresponds to some $u_j \in U$.

Since the graph G has girth at least 5, no vertex in D can lie in the set $\{u_j\} \cup N(u_j)$ and hence must be untouched by the application of P_1. Since u_j is a neighbor of v, it is clear that the application of P_1 leaves any vertex disjoint from $N_2[v]$ untouched. Now, suppose that after the application of P_1, a vertex w disjoint from $N_2[v] \setminus D$ has lost a degree. Then, it must be the case that the application of P_1 identified two of w's neighbors, say w_1 and w_2 as the vertex z_1. But since P_1 is applied on the vertex u_j, this implies the existence of a 4 cycle u_j, w_1, w, w_2 in G, which is a contradiction.

We assume as the induction hypothesis that the claim holds for all i' such that $1 \leq i' < i$ for some $i > 1$. Now, consider the application of P_i. By Lemma 6.4.10, this application cannot be on any of the vertices created by the application of P_l ($l < i$), and by the induction hypothesis, after the application of P_{i-1}, any vertex disjoint from $N_2[v] \setminus D$ remains untouched and retains the degree (which is ≥ 3) it had in the original graph. Hence, the application of P_i must occur on some vertex $u_j \in U$. Now, suppose that a vertex w disjoint from $N_2[v] \setminus D$ has lost a degree. Then, it must be the case that P_i identified two of w's neighbors say w_1 and w_2 as the vertex z_i. Since P_i is applied on the vertex u_j, this implies the existence of a 4 cycle u_j, w_1, w, w_2 in G, which is a contradiction. Finally, after the application of P_i, since no vertex outside $N_2[v] \setminus D$ has ever lost degree and they all had degree at least 3 to begin with, we cannot apply Preprocessing Rule 3 any further. This completes the proof of the claim.

Hence, after applying Preprocessing Rule 3 exhaustively on $G \setminus \{v\}$, any vertex disjoint from $N_2[v]$ is untouched and has the same degree as in the graph G. This completes the proof of the lemma. $\square$

Recall that the graph is irreducible if none of the preprocessing rules or Branching Rules **B**1 through **B**5 apply, i.e: the algorithm has reached **B**6.

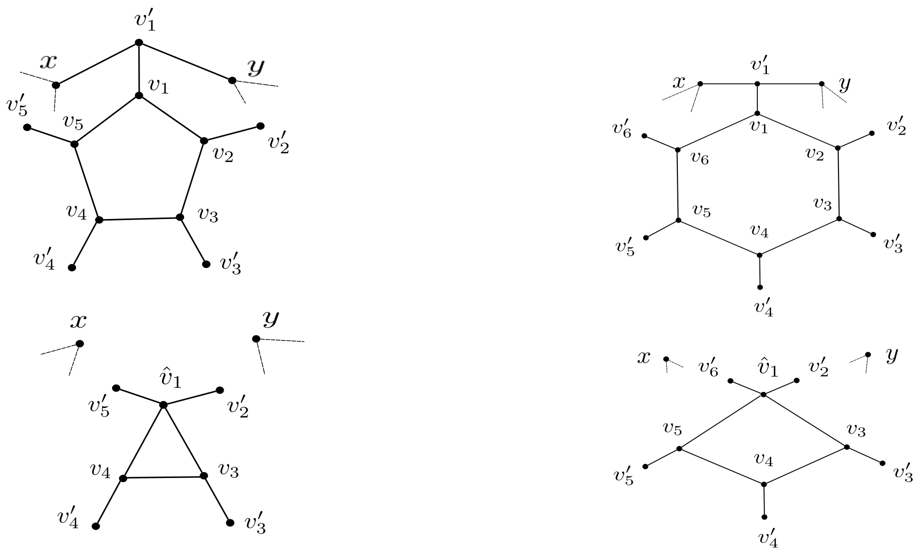

Figure 6.9: Cases of Lemma 6.4.13 when there is a 5 cycle or a 6 cycle in the graph

Lemma 6.4.13. *Let G be a connected 3-regular irreducible graph with at least 11 vertices. Then, $g(G) \geq 7$.*

Proof. 1. Suppose G contains a triangle v_1, v_2, v_3. Let v_4 be the remaining neighbor of v_1. Now, $G[N(v_1) \setminus \{v_4\}]$ is a clique, which implies that Branching Rule **B**2 applies and hence contradicts the irreducibilty of G. Hence, $g(G) \geq 4$.

2. Suppose G contains a cycle v_1, v_2, v_3, v_4 of length 4. Since G does not contain triangles, it must be the case that v_1 and v_3 are independent. Recall that G has minimum surplus 2, and hence surplus of the set $\{v_1, v_3\}$ is at least 2. Since v_2 and v_4 account for two neighbors of both v_1 and v_3, the neighborhood of $\{v_1, v_3\}$ can contain at most 2 more vertices (G is 3 regular). Since the minimum surplus of G is 2, $|N(\{v_1, v_3\})| = 4$ and hence $\{v_1, v_3\}$ is a minimum surplus set of size 2, which implies that Branching Rule **B**1 applies and hence contradicts the irreducibility of G. Hence, $g(G) \geq 5$.

3. Suppose that G contains a 5 cycle $v_1, \dots, v_5$. Since $g(G) \geq 5$, this cycle does not contain chords. Let v'_i denote the remaining neighbor of the vertex v_i in the graph G. Since there are no triangles or 4 cycles, $v'_i \neq v'_j$ for any $i \neq j$, and for any i and j such that $|i - j| = 1$, v'_i and v'_j are independent. Now, we consider the following 2 cases.

   Case 1: Suppose that for every i, j such that $|i - j| \neq 1$, v'_i and v'_j are adjacent. Then, since G is a connected 3-regular graph, G has size 10, which is a contradiction.

   Case 2: Suppose that for some i, j such that $|i - j| \neq 1$, v'_i and v'_j are independent (see Figure 6.9). Assume without loss of generality that $i = 1$ and $j = 3$. Consider the vertex v'_1 and let x and y be the remaining 2 neighbors of v'_1 (the first neighbor being v_1). Note that x or y cannot be incident to v_3, since otherwise x or y will coincide with v'_3. Hence, v_3 is disjoint from $N_2[v'_1]$. By Lemma 6.4.10 and Lemma 6.4.12, only Preprocessing Rule 3 applies and the applications are only on the vertices v_1, x and y and leaves v_3 untouched and the degree of vertex v_3 unchanged. Now, let $\hat{v}_1$ be the vertex which is created as a result of applying Preprocessing Rule 3 on v_1. Let $\hat{v}_4$ be the vertex created when v_4 is identified with another vertex during some application of Preprocessing Rule 3. If v_4 is untouched, then we let $\hat{v}_4 = v_4$. Similarly, let $\hat{v}'_3$ be the vertex created when v'_3 is identified with another vertex during some application of Preprocessing Rule 3. If v'_3 is untouched, then we let $\hat{v}'_3 = v'_3$. Since v_3 is untouched and its degree remains 3 in the graph $\mathcal{R}(G \setminus \{v\})$, the neighborhood of v_3 in this graph can be covered by a 2 clique $\hat{v}_1, \hat{v}_4$ and a vertex $\hat{v}'_3$, which implies that Branching Rule **B**2 applies in this graph, implying that Branching Rule **B**5 applies in the graph G, contradicting the irreducibility of G. Hence, $g(G) \geq 6$.

4. Suppose that G contains a 6 cycle $v_1, \ldots, v_6$. Since $g(G) \geq 6$, this cycle does not contain chords. Let v'_i denote the remaining neighbor of each vertex v_i in the graph G. Let x and y denote the remaining neighbors of v'_1 (see Figure 6.9). Note that both v_3 and v_5 are disjoint from $N_2[v'_1]$ (if this were not the case, then we would have cycles of length ≤ 5). Hence, by Lemma 6.4.10 and Lemma 6.4.12, we know that only Preprocessing Rule 3 applies and the applications are only on the vertices v_1, x and y, vertices v_3 and v_5 are untouched, and the degree of v_3 and v_5 in the graph $\mathcal{R}(G \setminus \{v'_1\})$ is 3. Let $\hat{v}_1$ be the vertex which is created as a result of applying Preprocessing Rule 3 on v_1. Let $\hat{v}_4$ be the vertex created when v_4 is identified with another vertex during some application of Preprocessing Rule 3. If v_4 is untouched, then we let $\hat{v}_4 = v_4$. Now, in the graph $\mathcal{R}(G \setminus \{v'_1\})$, the vertices v_3 and v_5 are independent and share two neighbors $\hat{v}_1$ and $\hat{v}_4$. The fact that they have degree 3 each and the surplus of graph $\mathcal{R}(G \setminus \{v'_1\})$ is at least 2 (Lemma 6.4.10, Lemma 6.4.7) implies that $\{v_3, v_5\}$ is a minimum surplus set of size at least 2 in the graph $\mathcal{R}(G \setminus \{v'_1\})$, which implies that branching rule **B**2 applies in this graph, implying that Branching Rule **B**5 applies in the graph G, contradicting the irreducibility of G. Hence, $g(G) \geq 7$.

This completes the proof of the lemma. □

Correctness and Analysis of the last step

In this section we combine all the results proved above and show the existence of degree 4 vertices in subsequent branchings after **B**6. Towards this we prove the following lemma.

Lemma 6.4.14. *Let G be a connected 3 regular irreducible graph on at least 11 vertices. Then, for any vertex $v \in V$,*

1. *$\mathcal{R}(G \setminus \{v\})$ contains three degree 4 vertices, say w_1, w_2, w_3; and*

2. *for any w_i, $i \in \{1, 2, 3\}$, $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_i\})$ contains w_j, $i \neq j$ as a degree 4 vertex.*

Proof. 1. Let v_1, v_2, v_3 be the neighbors of v. Since G was irreducible, **B**1, **B**2, **B**3 do not apply on $\mathcal{R}(G \setminus \{v\})$ (else **B**5 would have applied on G). By Lemma 6.4.10 and Lemma 6.4.12, we know that only Preprocessing Rule 3 would have been applied to get $\mathcal{R}(G \setminus \{v\})$ from $G \setminus \{v\}$ and the applications are only on these three vertices v_1, v_2, v_3. Let w_1, w_2 and w_3 be the three vertices which are created as a result of applying Preprocessing Rule 3 on these three vertices respectively. We claim that the degree of each w_i in the resulting graph is 4. Suppose that the degree of w_j is at most 3 for some j. But this can happen only if there was an edge between two vertices which are at a distance of 2 from v, that is, a path of length 3 between w_i and w_j for some $i \neq j$. This implies the existence of a cycle of length 5 in G, which contradicts Lemma 6.4.13.

2. Note that, by Lemma 6.4.12, it is sufficient to show that w_i is disjoint from $N_2[w_j]$ for any $i \neq j$. Suppose that this is not the case and let w_i lie in $N_2[w_j]$. First, suppose that w_i lies in $N_2[w_j] \setminus N_1[w_j]$ and there is no w_k in $N_1[w_i]$. Let x be a common neighbor of w_i and w_j. This implies that, in G, x has paths of length 3 to v via w_i and via w_j, which implies the existence of a cycle of length at most 6, a contradiction. Now, suppose that w_i lies in $N_1[w_j]$. But this can happen only if there was an edge between two vertices which are at a distance of 2 from v. This implies the existence of a cycle of length 5 in G, contradicting Lemma 6.4.13.

□

The next lemma shows the correctness of deleting v_{yz} from the graph $\mathcal{R}(G \setminus \{x\})$ without branching.

Lemma 6.4.15. *Let G be a connected irreducible graph on at least 11 vertices, v be a vertex of degree 3, and x, y, z be the set of its neighbors. Then, $G \setminus \{x\}$ contains a vertex cover of size at most k which excludes v if and only if $\mathcal{R}(G \setminus \{x\})$ contains a vertex cover of size at most $k-3$ which contains v_{yz}, where v_{yz} is the vertex created in the graph $G \setminus \{x\}$ by the application of Preprocessing Rule 3 on the vertex v.*

Proof. We know by Lemma 6.4.12 that there will be exactly 3 applications of Preprocessing Rule 3 in the graph $G \setminus \{x\}$, and they will be on the three neighbors of x. Let G_1, G_2, G_3 be the graphs which result after each such application, in that order. We assume without loss of generality that the third application of Preprocessing Rule 3 is on the vertex v.

By the correctness of Preprocessing Rule 3, if $G \setminus \{x\}$ has a vertex cover of size at most k which excludes v, then G_2 has a vertex cover of size at most $k-2$ which excludes v. Since this vertex cover must then contain y and z, it is easy to see that G_3 contains a vertex cover of size at most $k-3$ containing v_{yz}.

Conversely, if G_3 has a vertex cover of size at most $k-3$ containing v_{yz}, then replacing v_{yz} with the vertices y and z results in a vertex cover for G_2 of size at most $k-2$ containing y and z (by the correctness of Preprocessing Rule 3). Again, by the correctness of Preprocessing Rule 3, it follows that $G \setminus \{x\}$ contains a vertex cover of size at most k containing y and z. Since v is adjacent to only y and z in $G \setminus \{x\}$, we may assume that this vertex cover excludes v. □

Thus, when Branching Rule **B**6 applies on the graph G, we know the following about the graph.

- G is a 3 regular graph. This follows from the fact that Preprocessing Rules 1, 2 and 3 and the Branching Rule **B**4 do not apply.

- $g(G) \geq 7$. This follows from Lemma 6.4.13.

Let v be an arbitrary vertex and x, y and z be the neighbors of v. Since G is irreducible, Lemma 6.4.14 implies that $\mathcal{R}(G \setminus \{x\})$ contains 3 degree 4 vertices, w_1, w_2 and w_3. We let v_{yz} be w_1. Lemma 6.4.14 also implies that for any i, the graph $\mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus \{w_i\})$ contains 2 degree 4 vertices. Since the vertex v_{yz} is one of the three degree 4 vertices, in the graph $\mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus v_{yz})$, the vertices w_2 and w_3 have degree 4 and one of the Branching Rules **B**1, or **B**2, or **B**3 or **B**4 will apply in this graph. Hence, we combine the execution of the rule **B**6 along with the subsequent execution of one of the rules **B**1, **B**2, **B**3 or **B**4 (see Fig. 6.10). To analyze the drops in the measure for the combined application of these rules, we consider each root to leaf path in the tree of Fig. 6.10 (b) and argue the drops in each path.

- Consider the subtree in which v is not picked in the vertex cover from G, that is, x is picked in the vertex cover, following which we branch on some vertex w during the subsequent branching, from the graph $\mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus v_{yz})$.

  Let the instances (corresponding to the nodes of the subtree) be (G, k), (G_1, k_1), (G_2, k_2) and (G_2', k_2'). That is, $G_1 = \mathcal{R}(\mathcal{R}(G \setminus \{x\}) \setminus \{v_{yz}\})$, $G_2' = \mathcal{R}(G_1 \setminus \{w\})$ and $G_2 = \mathcal{R}(G_1 \setminus N[w])$.

  By Lemma 6.4.1, we know that $\mu(G \setminus \{x\}, k-1) \leq \mu(G, k) - \frac{1}{2}$. This implies that $\mu(\mathcal{R}(G \setminus \{x\}), k') \leq \mu(G, k) - \frac{1}{2}$ where $(\mathcal{R}(G \setminus \{x\}), k')$ is the instance obtained by applying the preprocessing rules on $G \setminus \{x\}$.

  By Lemma 6.4.1, we also know that including v_{yz} into the vertex cover will give a further drop of $\frac{1}{2}$. Hence, $\mu(\mathcal{R}(G \setminus \{x\}) \setminus \{v_{yz}\}, k'-1) \leq \mu(G, k) - 1$. Applying further preprocessing will not increase the measure. Hence $\mu(G_1, k_1) \leq \mu(G, k) - 1$.

Now, when we branch on the vertex w in the next step, we know that we use one of the rules **B**1, **B**2, **B**3 or **B**4. Hence, $\mu(G_2, k_2) \leq \mu(G_1, k_1) - \frac{3}{2}$ and $\mu(G_2', k_2') \leq \mu(G_1, k_1) - \frac{1}{2}$ (since **B**4 gives the worst branching vector). But this implies that $\mu(G_2, k_2) \leq \mu(G, k) - \frac{5}{2}$ and $\mu(G_2', k_2') \leq \mu(G, k) - \frac{3}{2}$.

This completes the analysis of the branch of rule **B**6 where v is not included in the vertex cover.

- Consider the subtree in which v is included in the vertex cover, by Lemma 6.4.14 we have that $\mathcal{R}(G \setminus \{v\})$ has exactly three degree 4 vertices, say w_1, w_2, w_3 and furthermore for any w_i, $i \in \{1, 2, 3\}$, $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_i\})$ contains 2 degree 4 vertices. Since G is irreducible, we have that for any vertex v in G, the Branching Rules **B**1, **B**2 and **B**3 do not apply on the graph $\mathcal{R}(G \setminus \{v\})$. Thus, we know that in the branch where we include v in the vertex cover, the first branching rule that applies on the graph $\mathcal{R}(G \setminus \{v\})$ is **B**4. Without loss of generality, we assume that **B**4 is applied on the vertex w_1. Thus, in the branch where we include w_1 in the vertex cover, we know that $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_1\})$ contains w_2 and w_3 as degree 4 vertices, This implies that in the graph $\mathcal{R}(\mathcal{R}(G \setminus \{v\}) \setminus \{w_1\})$ one of the Branching Rules **B**1, **B**2, **B**3 or **B**4 apply on a vertex w_1^*. Hence, we combine the execution of the rule **B**6 along with the subsequent executions of **B**4 and one of the rules **B**1, **B**2, **B**3 or **B**4 (see Fig. 6.10).

  We let the instances corresponding to the nodes of this subtree be (G, k), (G_1, k_1), (G_2, k_2), (G_2', k_2'), (G_3, k_3) and (G_3', k_3'), where $G_1 = \mathcal{R}(G \setminus \{v\})$, $G_2 = \mathcal{R}(G_1 \setminus N[w_1])$, $G_2' = \mathcal{R}(G_1 \setminus \{w_1\})$, $G_3 = \mathcal{R}(G_2' \setminus N[w_1^*])$ and $G_3' = \mathcal{R}(G_2' \setminus \{w_1^*\})$.

  Lemma 6.4.1, and the fact that preprocessing rules do not increase the measure implies that $\mu(G_1, k_1) \leq \mu(G, k)$.

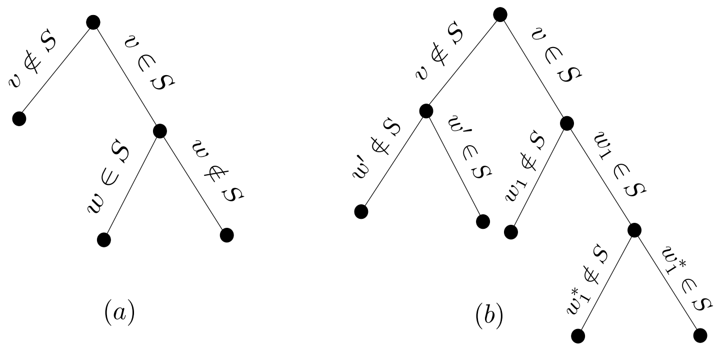

Figure 6.10: Illustrations of the branches of rules (a) **B**5 and (b) **B**6

Now, since **B**4 has been applied to branch on w_1, the analysis of the drop of measure due to **B**4 shows that $\mu(G_2, k_2) \leq \mu(G_1, k_1) - \frac{3}{2}$ and $\mu(G'_2, k'_2) \leq \mu(G_1, k_1) - \frac{1}{2}$. Similarly, since, in the graph G'_2, we branch on vertex w_1^* using one of the rules **B**1, **B**2, **B**3 or **B**4, we have that $\mu(G_3, k_3) \leq \mu(G'_2, k'_2) - \frac{3}{2}$ and $\mu(G'_3, k'_3) \leq \mu(G'_2, k'_2) - \frac{1}{2}$.

Combining these, we get that $\mu(G_3, k_3) \leq \mu(G, k) - \frac{5}{2}$ and $\mu(G'_3, k'_3) \leq \mu(G, k) - \frac{3}{2}$. This completes the analysis of rule **B**6 where v is included in the vertex cover. Combining the analysis for both the cases results in a branching vector of $(\frac{3}{2}, \frac{5}{2}, \frac{5}{2}, \frac{3}{2}, 2)$ for the rule **B**6.

Finally, we combine all the above results to obtain the following theorem.

Theorem 6.4.16. VERTEX COVER ABOVE LP *can be solved in time* $\mathcal{O}^*((2.3146)^{k-vc^*(G)})$.

Proof. Let us fix $\mu = \mu(G, k) = k - vc^*(G)$. We have thus shown that the preprocessing rules do not increase the measure. Branching Rules **B**1 or **B**2 or **B**3 results in a $(1, 1)$ decrease in $\mu(G, k) = \mu$, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + T(\mu - 1)$ which solves to $2^\mu = 2^{k - vc^*(G)}$.

Rule	**B**1	**B**2	**B**3	**B**4	**B**5	**B**6
Branching Vector	(1,1)	(1,1)	(1,1)	$(\frac{1}{2}, \frac{3}{2})$	$(\frac{3}{2}, \frac{3}{2}, 1)$	$(\frac{3}{2}, \frac{3}{2}, \frac{5}{2}, \frac{5}{2}, 2)$
Running time	2^μ	2^μ	2^μ	2.1479^μ	2.3146^μ	2.3146^μ

Figure 6.11: A table giving the decrease in the measure due to each branching rule.

Branching Rule **B**4 results in a $(\frac{1}{2}, \frac{3}{2})$ decrease in $\mu(G, k) = \mu$, resulting in the recurrence $T(\mu) \leq T(\mu - \frac{1}{2}) + T(\mu - \frac{3}{2})$ which solves to $2.1479^\mu = 2.1479^{k - vc^*(G)}$.

Branching Rule **B**5 combined with the next step in the algorithm results in a $(1, \frac{3}{2}, \frac{3}{2})$ branching vector, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + 2T(\mu - \frac{3}{2})$ which solves to $2.3146^\mu = 2.3146^{k - vc^*(G)}$.

We analyzed the way the algorithm works after an application of Branching Rule **B**6 before Theorem 6.4.16. An overview of drop in measure is given in Figure 6.4.6.

This leads to a $(\frac{3}{2}, \frac{5}{2}, 2, \frac{3}{2}, \frac{5}{2})$ branching vector, resulting in the recurrence $T(\mu) \leq T(\mu - 1) + 2T(\mu - \frac{3}{2})$ which solves to $2.3146^\mu = 2.3146^{k - vc^*(G)}$.

Thus, we get an $\mathcal{O}(2.3146^{(k - vc^*(G))} n^{O(1)})$ algorithm for VERTEX COVER ABOVE LP. $\square$

6.5 Applications

In this section we give several applications of the algorithm developed for VERTEX COVER ABOVE LP.

6.5.1 An algorithm for ABOVE GUARANTEE VERTEX COVER

Since the value of the LP relaxation is at least the size of the maximum matching, our algorithm also runs in time $\mathcal{O}^*(2.3146^{k-m})$ where k is the size of the minimum vertex cover and m is the size of the maximum matching.

Theorem 6.5.1. Above Guarantee Vertex Cover *can be solved in time* $\mathcal{O}^*(2.3146^\ell)$ *time, where* ℓ *is the excess of the minimum vertex cover size above the size of the maximum matching.*

Now by the known reductions in [44, 78, 88] (see also Figure 5.1) we get the following corollary to Theorem 6.5.1.

Corollary 6.5.2. Almost 2-SAT, Almost 2-SAT(v), RHorn-Backdoor Set Detection *can be solved in time* $\mathcal{O}^*(2.3146^k)$*, and* KVD$_{pm}$ *can be solved in time* $\mathcal{O}^*(2.3146^{\frac{k}{2}}) = \mathcal{O}^*(1.5214^k)$.

6.5.2 Algorithms for Odd Cycle Transversal and Split Vertex Deletion

We describe a generic algorithm for both Odd Cycle Transversal and Split Vertex Deletion. Let $X, Y \in \{\text{Clique, Independent Set}\}$. A graph G is called an (X, Y)*-graph* if its vertices can be partitioned into X and Y. Observe that when $X = Y =$ *independent set*, this corresponds to a *bipartite graph* and when $X =$ *clique* and $Y =$ *independent set*, this corresponds to a *split graph*. In this section we outline an algorithm that runs in time $\mathcal{O}^*(2.3146^k)$ and solves the following problem.

(X,Y)-Deletion Set **Parameter:** k

Input: An undirected graph G and a positive integer k

Question: Does G have a vertex subset S of size at most k such that its deletion leaves a (X, Y)-graph?

We solve the (X,Y)-DELETION SET problem by using a parameter preserving reduction to the ALMOST 2 SAT(V) problem.

Construction : Given a graph $G = (V, E)$ and (X, Y), we construct a 2-SAT formula $\phi(G, X, Y)$ as follows. The formula $\phi(G, X, Y)$ has a variable x_v for each vertex $v \in V$. We now add the following clauses to $\phi(G, X, Y)$. If $X =$ *clique*, then, for every non-edge $(u, v) \notin E$, we add the clause $(x_u \vee x_v)$. If $X =$ *independent set*, then for every edge $(u, v) \in E$, we add the clause $(x_u \vee x_v)$. Similarly, if $Y =$ *clique*, then for every non-edge $(u, v) \notin E$, we add the clause $(\bar{x}_u \vee \bar{x}_v)$ and if $Y =$ *independent set*, then for every edge $(u, v) \in E$, we add the clause $(\bar{x}_u \vee \bar{x}_v)$. This completes the construction of $\phi(G, X, Y)$.

Lemma 6.5.3. *Given a graph $G = (V, E)$ and (X, Y), let $\phi(G, X, Y)$ be the 2-SAT formula obtained by the above construction. Then, (G, k) is a* YES *instance of* (X,Y)-DELETION SET *if and only if $(\phi(G, X, Y), k)$ is a* YES *instance of* ALMOST 2 SAT(V).

Proof. Suppose there is a set $S \subseteq V$ such that $|S| \leq k$ and $G \setminus S$ is an (X, Y)-graph. Let S_v be the set of variables of $\phi = \phi(G, X, Y)$ which correspond to the vertices in S. Clearly, $|S_v| \leq k$. We claim that $\phi \setminus S_v$ is satisfiable by the following assignment. For each vertex in the X-partition of $G \setminus S$, assign the corresponding variable the value 0 and for each vertex in the Y-partition of $G \setminus S$, assign the corresponding variable the value 1. Suppose that this assignment does not satisfy $\phi \setminus S_v$ and let C be an unsatisfied clause. By the construction, we know that C is of the form $(x_u \vee x_v)$ or $(\bar{x}_u \vee \bar{x}_v)$. We consider only the first case, since the second is analogous to the first. If $(u, v) \in E$, then it must be the case that $X =$ *independent set* (by construction). Since this clause is unsatisfied, the value assigned to both x_u and x_v was 0. But this implies that u and v lie in the X-partition of $G \setminus S$, where $X =$ *independent set*, which is a contradiction. Similarly, if

$(u, v) \notin E$, then it must be the case that $X =$ *clique* (by construction). Since this clause is unsatisfied, the value assigned to both x_u and x_v was 0. But this implies that u and v lie in the X-partition of $G \setminus S$, where $X =$ *clique*, which is a contradiction.

Conversely, let S_v be a set of variables of $\phi = \phi(G, X, Y)$ such that $|S_v| \leq k$ and $\phi \setminus S_v$ is satisfiable. Let ρ be a satisfying assignment to $\phi \setminus S_v$ and let S be the set of vertices of G which correspond to S_v. Clearly, $|S| \leq k$. We now define the following partition of the vertices in $G \setminus S$. For each vertex of $G \setminus S$, if the corresponding variable is assigned 0 by ρ, then add it into partition A or into partition B otherwise. We claim that the partition (A, B) of $G \setminus S$ is an (X, Y) partition. Suppose that A is not an X-partition, where $X =$ *clique*. We only consider this case since the remaining cases can be argued analogously. Consider a non-edge (u, v) such that $u, v \in A$. But, by the construction, ϕ contains the clause $(x_u \vee x_v)$. Since $G \setminus S$ contains both the vertices u and v, it must be the case that $\phi \setminus S_v$ contains both x_u and x_v, implying that it contains the clause $(x_u \vee x_v)$. But, by the construction of the set A, ρ assigned 0 to both x_u and x_v, which is a contradiction. This completes the proof of the lemma. □

Combining the above lemma with Theorem 6.5.1, we have the following.

Theorem 6.5.4. (X,Y)-DELETION SET *can be solved in time* $\mathcal{O}^*(2.3146^k)$.

As a corollary to the above theorem we get the following new results.

Corollary 6.5.5. ODD CYCLE TRANSVERSAL *and* SPLIT VERTEX DELETION *can be solved in time* $\mathcal{O}^*(2.3146^k)$.

Observe that the reduction from EDGE BIPARTIZATION to ODD CYCLE TRANSVERSAL represented in Figure 5.1, along with the above corollary implies that EDGE BIPARTIZATION can also be solved in time $\mathcal{O}^*(2.3146^k)$. However, we note that there is an algorithm for this problem due to Guo et al. [46], running in time $\mathcal{O}^*(2^k)$.

6.5.3 An algorithm for KÖNIG VERTEX DELETION

A graph G is called König if the size of a minimum vertex cover equals that of a maximum matching in the graph. Clearly bipartite graphs are König but there are non-bipartite graphs that are König (a triangle with an edge attached to one of its vertices, for example). Thus the KÖNIG VERTEX DELETION problem, as stated below, is closely connected to ODD CYCLE TRANSVERSAL.

KÖNIG VERTEX DELETION (KVD) **Parameter:** k
Input: An undirected graph G and a positive integer k
Question: Does G have a vertex subset S of size at most k such that $G \setminus S$ is a König graph?

If the input graph G to KÖNIG VERTEX DELETION has a perfect matching then this problem is called KVD$_{pm}$. By Corollary 6.5.2, we already know that KVD$_{pm}$ has an algorithm with running time $\mathcal{O}^*(1.5214^k)$ by a polynomial time reduction to ABOVE GUARANTEE VERTEX COVER, that maps k to $k/2$. However, there is no known reduction if we do not assume that the input graph has a perfect matching and it required several interesting structural theorems in [81] to show that KVD can be solved as fast as ABOVE GUARANTEE VERTEX COVER. Here, we outline an algorithm for KVD that runs in $\mathcal{O}^*(1.5214^k)$ and uses an interesting reduction rule. However, for our algorithm we take a detour and solve a slightly different, although equally interesting problem. Given a graph, a set S of vertices is called *König vertex deletion set (kvd set)* if its removal leaves a König graph. The auxiliary problem we study is following.

VERTEX COVER PARAM BY KVD **Parameter:** k

Input: An undirected graph G, a König vertex deletion set S of size at most k and a positive integer ℓ

Question: Does G have a vertex cover of size at most ℓ?

This fits into the recent study of problems parameterized by other structural parameters. See, for example, ODD CYCLE TRANSVERSAL parameterized by various structural parameters [57] or TREEWIDTH parameterized by vertex cover [7] or VERTEX COVER parameterized by feedback vertex set [56] or DOMINATING SET parameterized by max-leaf number [30]. For our proofs we will use the following characterization of König graphs.

Lemma 6.5.6. [81, Lemma 1] *A graph $G = (V, E)$ is König if and only if there exists a bipartition of V into $V_1 \uplus V_2$, with V_1 a vertex cover of G such that there exists a matching across the cut (V_1, V_2) saturating every vertex of V_1.*

Note that in VERTEX COVER PARAM BY KVD, $G \backslash S$ is a König graph. So one could branch on all subsets of S to include in the output vertex cover, and for those elements not picked in S, we could pick its neighbors in $G \setminus S$ and delete them. However, the resulting graph need not be König adding to the complications. Note, however, that such an algorithm would yield an $\mathcal{O}^*(2^k)$ algorithm for VERTEX COVER PARAM BY OCT. That is, if S were an odd cycle transversal then the resulting graph after deleting the neighbors of vertices not picked from S will remain a bipartite graph, where an optimum vertex cover can be found in polynomial time.

Given a graph $G = (V, E)$ and two disjoint vertex subsets V_1, V_2 of V, we let (V_1, V_2) denote the bipartite graph with vertex set $V_1 \cup V_2$ and the edge set described as $\{\{u, v\} : \{u, v\} \in E \text{ and } u \in V_1, v \in V_2\}$. Now, we describe an algorithm based on Theorem 6.3.8, that solves VERTEX COVER PARAM BY KVD in time $\mathcal{O}^*(1.5214^k)$.

Theorem 6.5.7. Vertex Cover Param by KVD *can be solved in time* $\mathcal{O}^*(1.5214^k)$.

Proof. Let G be the input graph, S be a kvd set of size at most k. We first apply Lemma 6.3.1 on $G = (V, E)$ and obtain an optimum solution to LPVC(G) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V^x_{1/2}]$). Due to Lemma 6.3.2, this implies that there exists a minimum vertex cover of G that contains all the vertices in V^x_1 and none of the vertices in V^x_0. Hence, the problem reduces to finding a vertex cover of size $\ell' = \ell - |V^x_1|$ for the graph $G' = G[V^x_{1/2}]$. Before we describe the rest of the algorithm, we prove the following lemma regarding kvd sets in G and G' which shows that if G has a kvd set of size at most k then so does G'. Even though this looks straight forward, the fact that König graphs are not hereditary (i.e. induced subgraphs of König graphs need not be König) makes this a non-trivial claim to prove.

Lemma 6.5.8. *Let G and G' be defined as above. Let S be a kvd set of graph G of size at most k. Then, there is a kvd set of graph G' of size at most k.*

Proof. It is known that the sets $(V^x_0, V^x_1, V^x_{1/2})$ form a *crown decomposition* of the graph G [20]. In other words, $N(V^x_0) = V^x_1$ and there is a matching saturating V^x_1 in the bipartite graph (V^x_1, V^x_0). The set V^x_0 is called the *crown* and the set V^x_1 is called the *head* of the decomposition. For ease of presentation, we will refer to the set V^x_0 as C, V^x_1 as H and the set $V^x_{1/2}$ as R. In accordance with Lemma 6.5.6, let A be the minimum vertex cover and let I be the corresponding independent set of $G \setminus S$ such that there is a matching saturating A across the bipartite graph (A, I). First of all, note that if the set S is disjoint from $C \cup H$, $H \subseteq A$, and $C \subseteq I$, we are done, since the set S itself can be taken as a kvd set for G'. This last assertion follows because there exists a matching saturating H into C. Hence, we may assume that this is not the case. However, we will argue that given a kvd set of G of size at most k we will always be able to modify it in a way that it is of size at most k, it is disjoint from $C \cup H$, $H \subseteq A$, and $C \subseteq I$. This will

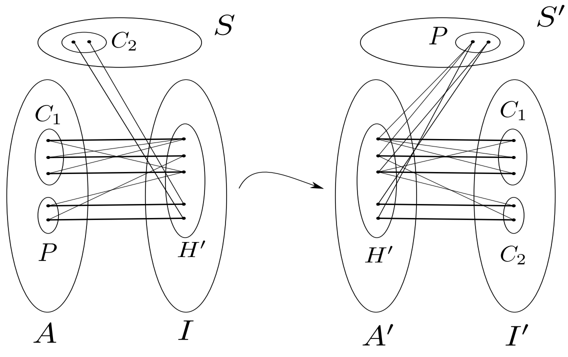

Figure 6.12: An illustration of case 2 of Lemma 6.5.8

allow us to prove our lemma. Towards this, we now consider the set $H' = H \cap I$ and consider the following two cases.

1. H' is empty. We now consider the set $S' = S \setminus (C \cup H)$ and claim that S' is also a kvd set of G of size at most k such that $G \setminus S'$ has a vertex cover $A' = (A \setminus C) \cup H$ with the corresponding independent set being $I' = I \cup C$. In other words, we move all the vertices of H to A and the vertices of C to I. Clearly, the size of the set S' is at most that of S. The set I' is independent since I was initially independent, and the newly added vertices have edges only to vertices of H, which are not in I'. Hence, the set A' is indeed a vertex cover of $G \setminus S'$. Now, the vertices of R, which lie in A, (and hence A') were saturated by vertices not in H, since $H \cap I$ was empty. Hence, we may retain the matching edges saturating these vertices, and as for the vertices of H, we may use the matching edges given by the crown decomposition to saturate these vertices and thus there is a matching saturating every vertex in A' across the bipartite graph (A', I'). Hence, we now have a kvd set S' disjoint from $C \cup H$, such that H is part of the vertex cover and C lies in the independent set of the König graph $G \setminus S'$.

2. H' is non empty. Let C_1 be the set of vertices in $A \cap C$ which are adjacent to H' (see Fig. 6.5.8) , let C_2 be the set of vertices in $C \cap S$, which are adjacent to H', and let P be the set of vertices of $R \cap A$ which are saturated by vertices of H' in the bipartite graph (A, I). We now consider the set $S' = (S \setminus C_2) \cup P$ and claim that S' is also a kvd set of G of size at most k such that $G \setminus S'$ has a minimum vertex cover $A' = (A \setminus (C_1 \cup P)) \cup H'$ with the corresponding independent set being $I' = (I \setminus H') \cup (C_1 \cup C_2)$. In other words, we move the set H' to A, the sets C_1 and C_2 to I and the set P to S. The set I' is independent since I was independent and the vertices added to I are adjacent only to vertices of H, which are not in I'. Hence, A' is indeed a vertex cover of $G \setminus S'$. To see that there is still a matching saturating A' into I', note that any vertex previously saturated by a vertex not in H can still be saturated by the same vertex. As for vertices of H', which have been newly added to A, they can be saturated by the vertices in $C_1 \cup C_2$. Observe that $C_1 \cup C_2$ is precisely the neighborhood of H' in C and since there is a matching saturating H in the bipartite graph (H, C) by Hall's Matching Theorem we have that for every subset $\hat{H} \subseteq H$, $|N(\hat{H}) \cap (C_1 \cup C_2)| \geq |\hat{H}|$. Hence, by Hall's Matching Theorem there is a matching saturating A' in the bipartite graph (A', I'). It now remains to show that $|S'| \leq k$.

   Since $N(H') = C_1 \cup C_2$ in the bipartite graph (C, H), we know that $|C_1| + |C_2| \geq |H'|$. In addition, the vertices of C_1 have to be saturated in the bipartite graph (A, I) by vertices in H'. Hence, we also have that $|C_1| + |P| \leq |H'|$. This implies that $|C_2| \geq |P|$. Hence, $|S'| \leq |S| \leq k$. This completes the proof of the claim. But now, notice that we have a kvd set of size at most k such that there are no vertices of H in the independent set side of the corresponding König graph. Thus, we have fallen into Case 1, which has been handled above.

This completes the proof of the lemma. $\square$

We now show that $\mu = vc(G') - vc^*(G') \leq \frac{k}{2}$. Let O be a kvd set of G' and define G'' as the Kónig graph $G' \setminus O$. It is well known that in König graphs, $|M| = vc(G'') = vc^*(G'')$, where M is a maximum matching in the graph G''. This implies that $vc(G') \leq vc(G'') + |O| = |M| + |O|$. But, we also know that $vc^*(G') \geq |M| + \frac{1}{2}(|O|)$ and hence, $vc(G') - vc^*(G') \leq \frac{1}{2}(|O|)$. By Lemma 6.5.8, we know that there is an O such that $|O| \leq k$ and hence, $vc(G') - vc^*(G') \leq \frac{k}{2}$.

By Corollary 6.3.9, in time $\mathcal{O}^*(2.3146^{vc(G')-vc^*(G')})$, we can compute a minimum vertex cover of G' and hence in time $\mathcal{O}^*(2.3146^{k/2})$. If the size of the minimum vertex cover obtained for G' is at most ℓ', then we return yes else we return no. We complete the proof of the theorem with a remark that, in the algorithm described above, we do not, in fact, even require a kvd set to be part of the input. $\square$

It is known that, given a minimum vertex cover, a minimum sized kvd set can be computed in polynomial time [81]. Hence, Theorem 6.5.7 has the following corollary.

Corollary 6.5.9. KVD *can be solved in time* $\mathcal{O}^*(1.5214^k)$.

Since the size of a minimum Odd Cycle Transversal is at least the size of a minimum König Vertex Deletion set, we also have the following corollary.

Corollary 6.5.10. VERTEX COVER PARAM BY OCT *can be solved in time* $\mathcal{O}^*(1.5214^k)$.

6.5.4 A simple improved kernel for VERTEX COVER

We give a kernelization for VERTEX COVER based on Theorem 6.3.8 as follows. Exhaustively, apply the Preprocessing rules 1 through 3 (see Section 6.3). When the rules no longer apply, if $k - vc^*(G) \leq \log k$, then solve the problem in time $\mathcal{O}^*(2.3146^{\log k}) =$

$\mathcal{O}(n^{\mathcal{O}(1)})$. Otherwise, just return the instance. We claim that the number of vertices in the returned instance is at most $2k - 2\log k$. Since $k - vc^*(G) > \log k$, $vc^*(G)$ is upper bounded by $k - \log k$. But, we also know that when Preprocessing Rule 1 is no longer applicable, all $\frac{1}{2}$ is the unique optimum to LPVC(G) and hence, the number of vertices in the graph G is twice the value of the optimum value of LPVC(G). Hence, $|V| = 2vc^*(G) \leq 2(k - \log k)$. Observe that by the same method we can also show that in the reduced instance the number of vertices is upper bounded by $2k - c\log k$ for any fixed constant c. Independently, Lampis [65] has also shown an upper bound of $2k - c\log k$ on the size of a kernel for VERTEX COVER for any fixed constant c.

6.6 Conclusion

We have demonstrated that using the change in LP values to analyze branching algorithms can give powerful results for parameterized complexity. We believe that our algorithm is the beginning of a race to improve the running time bound for ABOVE GUARANTEE VERTEX COVER. Furthermore, the running time bound for the classical VERTEX COVER problem, has seen no improvement in the last several years after an initial plethora of results. We believe that our algorithm may lead towards an improvement in this time bound by reducing the need to resort to too many refined branchings, which is possibly the reason why the progress in this direction has stagnated.

Our other contribution is to exhibit several parameterized problems that are equivalent to or reduce to ABOVE GUARANTEE VERTEX COVER through parameterized reductions.

Part III

Parity Based Graph Separation

7 Parity Multiway Cut

7.1 Introduction

In this chapter, we study a parity based generalization of the classical MULTIWAY CUT problem. Formally, we study the PARITY MULTIWAY CUT problem which is defined as:

PARITY MULTIWAY CUT (PMWC) **Parameter:** k

Input: A graph $G = (V, E)$, vertex subsets T_e and T_o ($T = T_e \cup T_o$), integer k

Question: Is there a vertex set S of size at most k which intersects

1. all odd paths from a vertex $v \in T_o$ to every other vertex $u \in T \setminus \{v\}$,
2. all even paths from a vertex $v \in T_e$ to every other vertex $u \in T \setminus \{v\}$?

When $T_e = T_o$, this is precisely the classical MULTIWAY CUT problem. If $T_o = \emptyset$ then this is the EVEN MULTIWAY CUT problem and if $T_e = \emptyset$ then this is the ODD MULTIWAY CUT problem.

Unlike MULTIWAY CUT, PMWC (when the solution is not allowed to contain the terminals) is already NP-complete for the case when $|T| = 2$. Indeed, consider the following reduction from VERTEX COVER to PMWC. Given an instance $(G = (V, E), k)$ of VERTEX COVER, add two new vertices t_1 and t_2, make them both adjacent to every vertex in V, and set $T_o = \{t_1, t_2\}$ and $T_e = \emptyset$. Call this new graph G'. It is easy to see that G has a vertex cover of size at most k if and only if G' has k-sized vertex subset that intersects every odd T_o-path. In fact, our argument shows that OMWC is NP-complete for the case when $|T| = 2$. One can similarly show that EMWC is NP-complete for the case when $|T| = 2$.

In this chapter, we study the variant of the problem where the solution to an instance $(G, T = T_e \cup T_o, k)$ of PMWC is allowed to intersect the set T. This problem is NP-complete for unbounded T because it is a generalization of MULTIWAY CUT. In this chapter, the focus is on optimizing the dependence of our algorithms on the parameter k. Therefore, we will suppress the dependence of the algorithms on the input size by using the $\mathcal{O}^*()$ notation. Our main result with respect to this problem is the following.

Theorem 7.1.1. PMWC *can be solved in time* $\mathcal{O}^*(2^{\mathcal{O}(k^3)})$.

In the course of obtaining our algorithm, we introduce two generic subroutines which apply to a wider range of graph separation problems. In particular,

- we first design an FPT algorithm that reduces the input instance to a highly structured instance while preserving a solution. In this particular case, the structure induced on the given instance is *bipartiteness* of the input graph. This algorithm gives a general approach for parity based graph separation problems and has already been utilized for other problems as well.

- we introduce a generalization of important separators which allows us to "overload" certain properties onto the parts of the graph disjoint from the separator.

The motivation behind such a generalization is that it allows us to define a more involved notion of **dominating set** of separators and therefore paves the way for the application of the Important Separators Template.

Overview of the algorithm. The algorithm for PMWC has three stages; in the first stage, using the technique of iterative compression, we reduce the problem to solving a bounded number of instances of the problem where the even terminals are bounded by a linear function of k. In the second stage, we reduce the instance to one with a solution whose removal leaves a bipartite graph. In the final stage, we define the generalization of important separators and apply the important separator template to solve this restricted version of the problem.

We note that the special case OMWC can be shown to be FPT be a reduction to the SUBSET ODD CYCLE TRANSVERSAL problem which was shown to be FPT in [58]. However, such an algorithm for OMWC would have a significantly worse dependence on the parameter k when compared to the algorithm we present in this chapter.

Our main motivation for studying this parity based generalizations of graph separation problems is the recently initiated parameterized study of parity versions of graphs minors by Kawarabayashi, Reed and Wollan [55] and separation problems similar to MULTIWAY CUT [10, 19, 79]. Furthermore, Geelen et al. [40] proved an odd variant of Mader's T-path Theorem in which they showed that, given a graph G and a subset T of vertices, there are either k vertex disjoint odd T-paths, or there is a vertex set of size at most $2k$ which intersects every odd T-path. This result has already turned out to be useful in graph theory [40, 59], as well as in the design of parameterized algorithms [52, 54, 53]. This result was crucial in settling the parameterized complexity of finding k vertex disjoint odd length cycles in a graph [53]. Observe that, this odd variant of Mader's T-path Theorem naturally gives rise to the OMWC problem, which is a special case of PMWC.

In an instance $(G, T = T_e \cup T_o, k)$ of PMWC, the vertices in T are called terminals, those in T_e are called *even terminals* and those in T_o are called *odd terminals*. Vertices in $T_e \setminus T_o$ are called *purely even terminals* and those in $T_o \setminus T_e$ are called *purely odd terminals*.

7.2 Bounding the number of even terminals

In this section, we give an algorithm which allows us to assume that the number of even terminals in a given instance is bounded linearly in k.

7.2.1 Preprocessing

We begin with a preprocessing rule which allows us to assume that the solution we are searching for is disjoint from the set of terminals.

Preprocessing Rule 1. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let $T = \{t_1, \ldots, t_l\}$. For every terminal $t_i \in T$, add 2 new vertices t_i^1 and t_i^2 and add edges (t_1, t_i^1) and (t_i^1, t_i^2) . Let this graph be G'. Set $T_o' = \{t_i^2 | t_i \in T_o\}$ and $T_e' = \{t_i^2 | t_i \in T_e\}$ and $T' = T_o' \cup T_e'$. Finally, return the instance $(G', T' = T_e' \cup T_o', k)$.*

It is clear that the above rule can be applied in polynomial time. The correctness of the rule follows from the following lemma.

Lemma 7.2.1. *Given an instance $(G, T = T_e \cup T_o, k)$ of* PMWC*, let $(G', T' = T_e' \cup T_o', k)$ be the instance obtained by the application of Preprocessing Rule 1 on $(G, T = T_e \cup T_o, k)$. Then, $(G, T = T_e \cup T_o, k)$ is a* YES *instance if and only if $(G', T' = T_e' \cup T_o', k)$ is a* YES *instance and has a solution disjoint from T'.*

Proof. Suppose that S is a solution for the instance $(G, T = T_e \cup T_o, k)$. We claim that S is a solution for the instance $(G', T' = T_e \cup T'_o, k)$ disjoint from T'. Clearly, $|S| \leq k$ and S is disjoint from T'. If S were not a solution for the instance $(G', T' = T'_e \cup T'_o, k)$ then there is a path P of forbidden parity between two terminals t_i^2 and t_j^2. But the subpath of P from t_i to t_j is a path with the same parity as P from t_i to t_j in $G \setminus S$, a contradiction.

Conversely, suppose that S' is a solution for the instance (G', T', k) disjoint from T'. If S' contains a vertex not in G, then it must be a vertex t_i^1 for some i. For every i such that $t_i^1 \in S'$, we replace it with the vertex t_i. Let the resulting set be S. Now, the vertices in S are all present in G. We claim that S is a solution for the instance $(G, T = T_e \cup T_o, k)$. Clearly, $|S| \leq k$. Therefore, if S were not a solution, then there is a path P of forbidden parity between two terminals t_i and t_j in the graph $G \setminus S$. Since t_i and t_j are not in S, t_i^1 and t_j^1 are not in S'. Let P_1 be the path comprising two edges from t_i^2 to t_i, that is, $P_1 = t_i^2, t_i^1, t_i$ and let P_2 be the path comprising two edges from t_j to t_j^2, that is, $P_2 = t_j, t_j^1, t_j^2$. Since P is a path in G, the paths P and P_1 intersect exactly in t_i and the paths P and P_2 intersect exactly in t_j. Therefore, the path $P_1 + P + P_2$ is a path from t_i^2 to t_j^2 in $G' \setminus S'$. However, since the paths P_1 and P_2 are of length 2, the path $P_1 + P + P_2$ is a path of the same parity as P from t_i^2 to t_j^2 in $G' \setminus S'$, a contradiction. This completes the proof of the lemma. $\square$

Due to the above preprocessing rule, we assume without loss of generality that the given instance of PMWC is such that each terminal has exactly 1 neighbor. Furthermore, we also assume that the objective is to check if there is a pmwc for the given instance disjoint from the terminal set. Formally, we redefine the PMWC problem in the following form.

PARITY MULTIWAY CUT (PMWC) **Parameter:** k

Input: A graph $G = (V, E)$, vertex subsets T_e and T_o ($T = T_e \cup T_o$), integer k

Question: Is there a vertex set S of size at most k which is disjoint from T and intersects

1. all odd paths from a vertex $v \in T_o$ to every other vertex $u \in T \setminus \{v\}$,
2. all even paths from a vertex $v \in T_e$ to every other vertex $u \in T \setminus \{v\}$?

Any set which hits all odd paths from a vertex $v \in T_o$ to every other vertex $u \in T \setminus \{v\}$ and all even paths from a vertex $v \in T_e$ to every other vertex $u \in T \setminus \{v\}$ is referred to as a *pmwc* for the given instance. We also initially perform the following preprocessing step on the given instance $(G, T = T_e \cup T_o, k)$ of PMWC.

Preprocessing Rule 2. *Given an instance* $(G, T = T_e \cup T_o, k)$ *of* PMWC, *if there is a vertex* v *which does not lie on a* T*-path, then return the instance* $(G \setminus \{v\}, T = T_e \cup T_o, k)$.

It is clear that the above rule is correct and can be applied in polynomial time by testing for every vertex if it lies on a T-path. Henceforth, we assume that Preprocessing Rule 2 is not applicable on the given input instance, which implies that every vertex lies on a T-path.

7.2.2 Reducing even terminals

We first describe a way to reduce the given instance of PMWC to multiple (but a bounded number of) instances, each with a bounded number of even terminals, such that solving these instances will lead to a solution for the input instance. To do this, we begin by using the technique of iterative compression.

Given an instance $(G = (V, E), T = T_e \cup T_o, k)$ of PMWC, where $V = \{v_1, \ldots, v_n\}$, we define the graph G_i as $G_i = G[V_i]$ where $V_i = \{v_1, \ldots, v_i\}$. We iterate through the instances $(G_i, T_i = (T_e \cap V_i) \cup (T_o \cap V_i), k)$ starting from $i = k + 1$ and for the i^{th} instance, with the help of a *known* solution S_i of size at most $k + 1$ we try to find a solution $\hat{S}_i$ of size at most k. Formally, the *compression* problem we address is following.

PMWC COMPRESSION **Parameter:** k

Input: An instance $(G, T = T_e \cup T_o, k)$ of PMWC, and a pmwc S of size at most $k + 1$.

Question: Is there a pmwc for the instance $(G, T = T_e \cup T_o, k)$ of size at most k?

We reduce the PMWC problem to $n - k$ instances of the PMWC COMPRESSION problem as follows. Let $I_i = (G_i, (T_e \cap V_i) \cup (T_o \cap V_i), S_i, k)$ be the i^{th} instance of PMWC COMPRESSION. The set V_{k+1} is clearly a pmwc of size at most $k + 1$ for the instance I_{k+1}. It is also easy to see that if $\hat{S}_{i-1}$ is a pmwc of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ (or the set $\hat{S}_{i-1} \cup N(v_i)$ if v_i is a terminal) is a pmwc of size at most $k + 1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+1}, (T_e \cap V_{k+1}) \cup (T_o \cap V_{k+1}), k, S_{k+1})$ where $S_{k+1} = V_{k+1}$ and check if there is a solution for this instance. If there is such a solution $\hat{S}_{k+1}$, we set $S_{k+2} = \hat{S}_{k+1} \cup \{v_{k+2}\}$ (or $S_{k+2} = \hat{S}_{k+1} \cup N(v_{k+2})$ if v_{k+1} is a terminal) and try to compute a pmwc of size at most k for the instance I_{k+2} and so on. If, during any iteration, the corresponding instance is found to be a NO instance then it implies that the original instance is also a NO instance. Finally the solution for the original input instance is the set $\hat{S}_n$. Since there can be at most n such iterations, the total time taken is bounded by

n times the time required to solve the PMWC COMPRESSION problem.

Observation 7.2.2. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let S be a solution for this instance.*
(a) *Consider a connected component of $G \setminus S$ which contains at least two terminals. Such a component cannot contain terminals from both T_o and T_e.*
(b) *Any connected component of $G \setminus S$ contains at most 2 vertices from T_e.*

Proof. **(a)** Let C be a component of $G \setminus S$ containing two terminals t_1 and t_2 such that $t_1 \in T_o$ and $t_2 \in T_e$. Then, there is a path between t_1 and t_2 in $G \setminus S$. If this path is odd, then it contradicts the intersection of S with every odd t_1-$T \setminus \{t_1\}$ path and if it is even, then it contradicts the intersection of S with every even t_2-$T \setminus \{t_2\}$ path.
(b) Let C be a component of $G \setminus S$ containing a set $T'_e \subseteq T_e$. Consider a tree spanning the set T'_e in $G \setminus S$. Since T'_e lies in a connected component of $G \setminus S$, such a tree exists. Since this tree is a connected bipartite graph, if $|T'_e| > 2$, there must be two vertices of T'_e with an even path between them. But this is a contradiction since this even path is disjoint from S. □

We now show that if an instance (G, T, k, S) of PMWC COMPRESSION has a solution disjoint from some pmwc S, then it must be the case that the number of even terminals in this instance is bounded linearly in k. Formally,

Lemma 7.2.3. *Consider an instance $(G, T = T_e \cup T_o, k, S)$ of* PMWC COMPRESSION *and suppose that this instance has a solution disjoint from the pmwc S. Then, $|T_e| \leq 6k$.*

Proof. Let S' be the solution disjoint from S. We call a component of $G \setminus S$ *affected* if it contains a vertex of S' and *unaffected* otherwise. Clearly, there can be at most k affected components in $G \setminus S$. Now, consider the unaffected components of $G \setminus S$ which contain an even terminal. We claim that the number of such components does not exceed $2k$. If this were not the case, then there exist three unaffected components which contain an even terminal each and a vertex $v \in S$ such that the three unaffected components are adjacent v. But this implies the presence of an even path between at least two of these even terminals which is disjoint from the solution S', a contradiction. Hence, the number of components of $G \setminus S$ which contain an even terminal is at most $3k$ (at most k affected and at most $2k$ unaffected components). By Observation 7.2.2, any component of $G \setminus S$ can contain at most 2 even terminals. Since S is disjoint from T_e, the number of even terminals in the instance is bounded by $2 \cdot 3k = 6k$. This completes the proof of the lemma. □

We now describe a way to bound the number of even terminals in an instance of PMWC COMPRESSION. Formally,

Lemma 7.2.4. *There is an algorithm that, given an instance $I = (G, T = T_e \cup T_o, k, S)$ of* PMWC COMPRESSION*, runs in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ and returns 2^{k+1} instances of* PMWC *$\{(G_i, T_i = T_{ei} \cup T_{oi}, k_i)\}_{1 \leq i \leq \ell}$ where $\ell = 2^{k+1}$ such that*

1. *(G, T, k, S) is a* YES *instance of* PMWC COMPRESSION *if and only if there is an $1 \leq i \leq \ell$ such that (G_i, T_i, k_i) is a* YES *instance of* PMWC.

2. *For each $1 \leq i \leq \ell$, $k_i \leq k$ and $|T_{ei}| \leq 6k_i$.*

Proof. For every $S' \subseteq S$, we obtain an instance $I_{S'}$ of PMWC COMPRESSION corresponding to S' by deleting S' from the instance I and applying Preprocessing Rule 2 exhaustively on the resulting instance. Thus, we obtain 2^{k+1} instances of PMWC COMPRESSION, each corresponding to a subset of S.

1. Suppose that (G, T, k, S) is a YES instance of PMWC COMPRESSION. We fix a hypothetical solution $\hat{S}$ for this instance. Let $Y = S \cap \hat{S}$ and consider the instance

$I_Y = (G \setminus Y, T \setminus Y, k - |Y|)$. Then, I_Y is a YES instance of PMWC and $\hat{S} \setminus Y$ is a solution for this instance which is also disjoint from $S \setminus Y$.

Conversely, if (G, T, k, S) were a NO instance, then it follows that for every $S' \subseteq S$, the corresponding instance $I_{S'} = (G \setminus S', T \setminus S', k - |S'|)$ of PMWC is also a NO instance.

2. Observe that if (G, T, k, S) is a YES instance and $Y = S \cap \hat{S}$ then the instance I_Y has a solution disjoint from $S \setminus Y$ and by Lemma 7.2.3, the number of even terminals in this instance cannot exceed $6(k - |Y|)$. Therefore, if for any instance $I_{S'}$, the number of even terminals in this instance exceeds $6(k - |S'|)$, then we term the instance invalid and reject it. Therefore, we may assume that at this juncture, we have at most 2^{k+1} instances of PMWC and the number of even terminals in each instance $I_{S'}$ is bounded by $6(k - |S'|)$. The algorithm finally returns these instances. We have already shown that these instances satisfy the properties in the statement of the lemma. Since each such instance can be constructed in polynomial time and there are at most 2^{k+1} such instances, the algorithm runs in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$. This completes the proof of the lemma. □

Henceforth, we assume that the given PMWC instance is one returned by the algorithm of Lemma 7.2.4.

7.3 Obtaining a bipartition instance

In this section, we give the first of our subroutines which imposes a certain structure on the given graph. This will be described and proved formally in the rest of this section.

7.3.1 Isolated and semi-isolated components

Definition 7.3.1. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC *and let S be a solution for this instance (see Fig. 7.1). A connected component of $G \setminus S$ is called an* **isolated component** *in $G \setminus S$ if it is disjoint from the set of terminals and a* **non-isolated component** *(in $G \setminus S$) otherwise. Vertices in an isolated component are called isolated vertices and those in a non-isolated component are called non-isolated vertices.*

Definition 7.3.2. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC *and let S be a solution for this instance. Consider a non-isolated component C in $G \setminus S$ which contains at least 2 terminals. We define a* **main component** *of C as a maximal connected subgraph of $G[C]$ such that every vertex in this subgraph lies on a T-path in $G \setminus S$. Consider a non-isolated component of C in $G \setminus S$ which contains a single terminal $t \in T$. We know that t has a single neighbor in G, say u. We define the main component of C as the edge (u, t).*

Observation 7.3.3. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC *and let S be a solution for this instance. Consider a non-isolated component C in $G \setminus S$. Then, there is a unique main component of C.*

Proof. If C contains a single terminal, it is clear that there is a unique main component by definition. Therefore, we assume that C contains at least 2 terminals. Suppose that there were two distinct main components of C, M_1 and M_2. Since each main component is a maximal connected subgraph, the two main components must be disjoint and non-adjacent. Furthermore, each main component contains a terminal. Since C is a connected component of $G \setminus S$, there is a path P in $G[C]$ from M_1 to M_2. Recall that P is not an edge. Since both main components contain terminals, there is path from a terminal in M_1 to a terminal in M_2 which contains P as a subpath. However, this implies that every vertex in P lies on a T-path in $G \setminus S$, which contradicts the maximality of M_1 and M_2. Hence, we conclude that there is a unique main component of C. $\square$

Definition 7.3.4. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC *and let S be a solution for this instance. Consider a non-isolated component C in $G \setminus S$ and let M be the main component of C. Then, the connected components of the graph $G[C \setminus M]$ are called the* **semi-isolated components** *of C in $G \setminus S$. Vertices in a semi-isolated component are called* **semi-isolated vertices**.

Observation 7.3.5. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC *and let S be a solution for this instance. Consider a non-isolated component C in $G \setminus S$ and let M be the main component of C in $G \setminus S$. Every terminal in C is also present in M. Furthermore, if C' is a semi-isolated component of C, then there is no T-path in $G \setminus S$ which intersects C'.*

Proof. Both statements are clearly true when C contains a single terminal. Therefore, we assume that C contains at least 2 terminals. Now, let $T' = T \cap C$ and consider a minimal tree, say H, in $G[C]$ which spans the terminals in T'. Since C is connected component, such a tree exists. Since H is minimal, every vertex in H lies on a T-path within H. Since the main component is unique, H is contained in the main component, which implies that all the vertices in T' occur in the main component of C. Furthermore, if there is a semi-isolated component C' such that there is a T'-path in C intersecting a vertex in C' then we may simply add the vertices of this T'-path to the main component to get another connected subgraph with every vertex lying on a T-path in $G \setminus S$. Since the main component is strictly contained inside this subgraph, it contradicts the maximality of the main component. $\square$

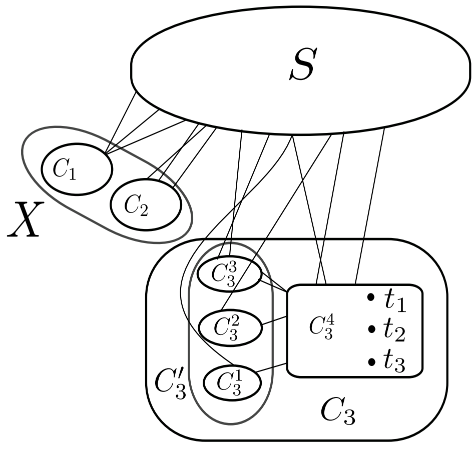

Figure 7.1: An illustration of a solution S where X is the isolated part of S, C_3 is a non-isolated component, C'_3 is the semi-isolated part of C_3 and C_3^4 is a main component.

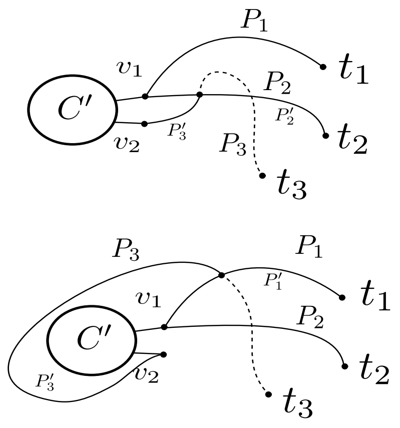

Figure 7.2: An illustration of the two subcases of case (b) in Lemma 7.3.6.

Lemma 7.3.6. *Let* $(G, T = T_e \cup T_o, k)$ *be an instance of* PMWC *and let* S *be a solution for this instance. Then, any semi-isolated component has exactly 1 neighbor in the adjacent main component.*

Proof. Consider a semi-isolated component C' and let M be the main component adjacent to C'. We will now show that C' cannot have more than one neighbor in M.

Suppose that this is not the case and let v_1 and v_2 be two distinct vertices in M which are adjacent to vertices in C'. We note that if v_1 and v_2 are terminals, then there is a path between these two terminals which intersects the semi-isolated component C', which is not possible by Observation 7.3.5. Now, consider the case when exactly one of the two vertices, say v_1, is a terminal. But, v_2 lies on a path between two terminals, say w_1 and w_2 where w_1 or w_2 could be v_1. Consider this path and consider the two subpaths of this path from v_2 to w_1 and from v_2 to w_2. At least one of these two subpaths is disjoint from v_1. Hence, this subpath, along with the edges from C' to v_1 and v_2 results in a T-path which intersects a semi-isolated component, a contradiction.

We now consider the remaining case where v_1,$v_2 \notin T$. Let P be a T-path from t_1 to t_2 which contains v_1. We know that such a path exists since v_1 lies in a main component. Let P_1 be the subpath of P from t_1 to v_1 and let P_2 be the subpath of P from v_1 to t_2. Let P_3 be a path from v_2 to $t_3 \in T$ (t_3 can be the same as t_1 or t_2). We know that such a path exists since v_2 is in a main component. We now consider the following two cases.

(a) P_3 does not intersect P_2 or P_1. Then clearly, there are paths from t_1 to t_3 and t_2 to t_3 which intersect the semi-isolated component C', and are disjoint from the solution, which is a contradiction since no vertex in C' can lie on a T-path disjoint from the solution.

(b) P_3 intersects P_2 or P_1. Without loss of generality, suppose that P_3 intersects P_2 first (see Fig. 7.2) when traversing from v_2 to t_3 and let the vertex at which this intersection occurs be u. Let P_3' be the subpath of P_3 from v_2 to u and let P_2' be the subpath of P_2 from u to t_2. Additionally, let P be a path from v_1 to v_2 such that the internal vertices of P lie in C'. Since C' is a connected component, we know that such a path exists. But now, $P_1 + P + P_3' + P_2'$ is a t_1-t_2 path disjoint from S and intersecting C'. This is a contradiction since no vertex in C' can lie on a T-path in $G \setminus S$ (by Observation 7.3.5).

This concludes the proof of the lemma. □

Definition 7.3.7. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let S be a solution for this instance. Let C be a non-isolated component of $G \setminus S$ and let C' be a semi-isolated component of C. We denote the unique neighbor of C' in the adjacent main component by $\chi(C')$ and refer to this vertex as the* **pivot** *of C'. For every vertex $v \in C'$, we define $\chi(v) = \chi(C')$.*

Lemma 7.3.8. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let S be a solution for this instance. Then, no vertex in T occurs as a pivot in $G \setminus S$.*

Proof. Suppose that a terminal $t \in T$ occurs as a pivot and let C be the component of $G \setminus S$ containing t. Recall that by assumption, every terminal in this instance has a single neighbor. Let u be the neighbor of t in G. Therefore, it must be case that u lies in a semi-isolated component of C. However, if C contains 2 terminals, then the main component of C is a connected subgraph containing both terminals (by Observation 7.3.5), which implies that the vertex u is also in the main component of C, a contradiction. Therefore, it must be the case that C does not contain a terminal other than t. But in this case, u is in the main component by definition, a contradiction. This completes the proof of the lemma. □

7.3.2 Branching on isolated and semi-isolated vertices

In this subsection, we show that given a an isolated or semi-isolated vertex (Lemma 7.3.10 and Lemma 7.3.12) with respect to some solution, we can compute a set of bounded size which intersects some solution for this instance.

Lemma 7.3.9. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC*. Let S be a solution for this instance and let v be an isolated vertex in $G \setminus S$. There is a solution for this instance which intersects an important v-T separator of size at most k.*

Proof. Due to Preprocessing Rule 2, v lies on a T-path in G. Since v is isolated in $G \setminus S$, it must be case that there is a non empty set $K \subseteq S$ such that K is a minimal v-T separator. If K is an important v-T separator, then S itself a solution of the required kind. Suppose that K is not an important v-T separator and let J be an important v-T separator dominating K. We claim that the set $S' = (S \setminus K) \cup J$ is also a solution for

```
    Input  : Instance $(G, T = T_e \cup T_o, k)$ of PMWC and a vertex $v$ such that $v$ is
             isolated with respect to some solution for the instance
    Output: A set $\mathcal{R}$ of vertices which intersects some solution
1   $\mathcal{X} \leftarrow$ set of important $v$-$T$ separators of size at most $k$
2   $\mathcal{R} \leftarrow \cup_{X \in \mathcal{X}} X$
3   return $\mathcal{R}$
```

Algorithm 7.3.1: Algorithm BRANCH-ISO to compute a set of vertices intersecting a solution when given a vertex isolated with respect to some solution.

the given instance. Clearly $|S'| \leq |S|$. Therefore, if S' were not a solution, then there is a T-path of forbidden parity in the graph $G \setminus S'$. Since S is a pmwc, this path must intersect $S \setminus S' = K \setminus J$. This implies the existence of a path from $K \setminus J$ to T in the graph $G \setminus S'$ which in turn implies the existence of a path from $K \setminus J$ to T in the graph $G \setminus J$ since $J \subseteq S'$.

However, since J dominates K (by Lemma 3.2.9), there is no path from $K \setminus J$ to T in $G \setminus J$, a contradiction. Therefore, S' is a solution for the given instance intersecting an important v-T separator of size at most k. This completes the proof of the lemma. □

Lemma 7.3.10. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC. *Let S be a solution for this instance and let v be an isolated vertex in $G \setminus S$. Given v, in time $\mathcal{O}^*(4^k)$, we can find a set of at most $4^k k$ vertices with a non-empty intersection with some solution for this instance.*

Proof. By Lemma 3.2.18 we know that the number of important v-T separators of size at most k is at most 4^k and these can be enumerated in time $\mathcal{O}^*(4^k)$. Therefore, to compute the required set intersecting a solution, we simply enumerate all important v-T separators of size at most k and return the set obtained by taking the union of the vertices in these separators . By Lemma 7.3.9, the returned set intersects some solution for the given instance. This completes the proof of the lemma. □

We now prove an analogous lemma for semi-isolated vertices.

Lemma 7.3.11. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC. *Let S be a solution for this instance and let v be a semi-isolated vertex in $G \setminus S$. There is a solution intersecting an important v-T separator of size at most $k + 1$ in the graph G.*

Proof. Due to Preprocessing Rule 2, v lies on a T-path in G. Since v is semi-isolated, it must be case that there is a non-empty set $K \subseteq S$ such that K is a minimal v-T separator in the graph $G \setminus \{\chi(v)\}$. If $K' = K \cup \{\chi(v)\}$ is an important v-T separator, then S itself is a solution of the required kind. Suppose that K' is not an important v-T separator and let J be an important v-T separator dominating K'. We now select a vertex u as follows. If $\chi(v) \in J$, then we set $u = \chi(v)$ and if $\chi(v) \notin J$, then we choose as u an arbitrary vertex in $J \setminus K$. We claim that the set $S' = (S \setminus K) \cup (J \setminus \{u\})$ is also a solution for the given instance. Clearly, $|S'| \leq |S|$. Therefore, if S' were not a solution, then there is a T-path intersecting $S \setminus S' = K \setminus J$. This implies that there is a vertex $z \in K \setminus J$ which has two vertex disjoint paths to T in $G \setminus (J \setminus \{u\})$. However, since J dominates K', J intersects all paths from z to T in G (by Lemma 3.2.9). Therefore, there cannot be 2 vertex disjoint paths from z to T in the graph $G \setminus (J \setminus \{u\})$, a contradiction. Hence, we conclude that S' is a solution for the given instance which intersects an important v-T separator of size at most $k + 1$. This completes the proof of the lemma. □

Lemma 7.3.12. *Consider an instance $(G, T = T_e \cup T_o, k)$ of* PMWC. *Let S be a solution for this instance and let v be a semi-isolated vertex with respect to S. Given v, in time $\mathcal{O}^*(4^k)$, we can find a set of at most $4^{k+1}(k + 1)$ vertices with a non-empty intersection with some solution for this instance.*

Proof. By Lemma 3.2.18 we know that the number of important X-Y separators of size at most $k + 1$ is at most 4^{k+1} and these can be enumerated in time $\mathcal{O}^*(4^k)$. Therefore,

```
Input  : Instance $(G, T = T_e \cup T_o, k)$ of PMWC and a vertex $v$ such that $v$ is
         semi-isolated with respect to some solution for the instance
Output: A set $\mathcal{R}$ of vertices which intersects some solution
1 $\mathcal{X} \leftarrow$ set of important $v$-$T$ separators of size at most $k+1$
2 $\mathcal{R} \leftarrow \cup_{X \in \mathcal{X}} X$
3 return $\mathcal{R}$
```

Algorithm 7.3.2: Algorithm BRANCH-SEMI-ISO to compute a set of vertices intersecting a solution when given a vertex semi-isolated with respect to some solution.

to compute the required set intersecting a solution, we simply enumerate all important v-T separators of size at most $k + 1$ and return the set obtained by taking the union of the vertices in these separators (see Algorithm 7.3.2). By Lemma 7.3.11, the returned set intersects some solution for the given instance. This completes the proof of the lemma. □

Definition 7.3.13. *Consider an instance $(G = (V, E), T, k)$ of* PMWC. *We call $J \subseteq V$ an* **important component** *if $G[J]$ is connected and $N(J)$ is an important J-T separator of size at most $k + 1$.*

Lemma 7.3.14. *Let $(G, T = T_e \cup T_o, k)$ be a* YES *instance of* PMWC. *Then, there is a solution S for this instance such that every every isolated or semi-isolated component in $G \setminus S$ is an important component.*

Proof. Consider the set of solutions for this instance with minimum size and among these let S be the solution which maximizes the number of vertices which are isolated or semi-isolated in $G \setminus S$. We claim that S is a solution which satisfies the statement of the lemma.

We first show that for any isolated or semi-isolated component C in $G \setminus S$, the set $N(C)$ is a minimal C-T separator. For this, it suffices to show that for any vertex

$v \in N(C)$, there is a C-T path in the graph $G \setminus (N(C) \setminus \{v\})$. Consider a vertex $v \in N(C)$. Suppose that C is a semi-isolated component and $v = \chi(C)$. There is a path from v to T in $G \setminus (N(C) \setminus \{v\})$ since v lies in a main component of $G \setminus S$ by definition and $N(C) \setminus \{v\}$ is contained in S. This implies the presence of a C-T path in $G \setminus (N(C) \setminus \{v\})$ since C is adjacent to v. Therefore, we may assume that if C is semi-isolated, then $v \neq \chi(C)$, which implies that $v \in S$. Now, since the solution S is minimal, there is a T-path containing v in $G \setminus (S \setminus \{v\})$, which implies that there are 2 vertex disjoint paths from v to T in $G \setminus (S \setminus \{v\})$ and hence there is a path from v to T in $G \setminus (N(C) \setminus \{v\})$. This implies the presence of a C-T path in $G \setminus (N(C) \setminus \{v\})$ since C is adjacent to v. Therefore, we conclude that $N(C)$ is indeed a minimal C-T separator.

For any isolated or semi-isolated component C, $|N(C)| \leq k + 1$. Therefore, it remains to show that the neighborhood of every isolated or semi-isolated component is an important separator.

Case 1: isolated components. Consider an isolated component C such that the set $X = N(C)$ is not an important C-T separator. Let Y be an important C-T separator dominating X. We claim that $S' = (S \setminus X) \cup Y$ is also a solution for the instance. Clearly, $|S'| \leq |S|$. Therefore, if S' were not a solution, then there is a T-path intersecting a vertex in $S \setminus S' = X \setminus Y$ in the graph $G \setminus S'$. This implies that there is a path from a vertex in $X \setminus Y$ to T in the graph $G \setminus Y$. But $X \setminus Y$ is separated from T by Y (by Lemma 3.2.9) leading to a contradiction. Therefore, S' is indeed a solution for this instance. Observe that if $|Y| < |X|$ or $(Y \setminus X) \cap S \neq \emptyset$, then S' is strictly smaller than S, which contradicts our choice of S. Therefore, we may assume that $|Y| = |X|$ and $(Y \setminus X) \cap S = \emptyset$. Let y be an arbitrary vertex in $Y \setminus X$. We claim that $S'' = S' \setminus \{y\}$ is also a solution for this instance. If this were not the case, then there is a T-path intersecting y in $G \setminus S''$. Since S is a solution, this path must intersect some vertex in $S \setminus S''$. Since $y \notin S$, it must be the case that this path intersects a vertex $z \in X \setminus Y$. Therefore, we have that there is a vertex $z \in X \setminus Y$ which has 2 vertex disjoint paths to T in $G \setminus S''$ and hence z has 2 vertex disjoint paths to T in $G \setminus (Y \setminus \{y\})$. However, we know that Y intersects all paths from z to T in G (by Lemma 3.2.9). Therefore, there cannot be 2 vertex disjoint paths from z to T in $G \setminus (Y \setminus \{y\})$, a contradiction. Hence, we conclude that S'' is also a solution for this instance. But S'' is strictly smaller than S', which in turn is no larger than S. Therefore, we have that S'' is a solution strictly smaller than S, which contradicts our choice of S. Hence, we conclude that $N(C)$ is indeed an important C-T separator.

Case 2: semi-isolated components. Consider a semi-isolated component C such that the set $X = N(C)$ is not an important C-T separator. Let Y be an important C-T separator dominating X. We select a vertex u as follows. If $\chi(C) \in Y$ then $u = \chi(C)$ and if $\chi(C) \notin Y$ then choose an arbitrary vertex in $Y \setminus X$ as u. We claim that $S' = (S \setminus X) \cup (Y \setminus \{u\})$ is also a solution for the instance. Clearly, $|S'| \leq |S|$. Therefore, if S' were not a solution, then there is a T-path intersecting a vertex z in $S \setminus S' = X \setminus Y$, implying that there are 2 vertex disjoint paths from z to T in $G \setminus (Y \setminus \{u\})$. But $X \setminus Y$ is separated from T by Y (by Lemma 3.2.9). Therefore, there cannot be 2 vertex disjoint paths from z to T in $G \setminus (Y \setminus \{u\})$, a contradiction. Therefore, S' is indeed a solution for this instance. Observe that if $|Y| < |X|$ or $(Y \setminus (X \cup \{u\})) \cap S \neq \emptyset$, then S' is strictly smaller than S, which contradicts our choice of S. Therefore, we may assume that $|Y| = |X|$ and $(Y \setminus (X \cup \{u\})) \cap S = \emptyset$.

We now prove the following properties regarding the set Y defined above.

Claim 6. **(a)** *Every vertex in $R(C,Y)$ is isolated or semi-isolated in $G \setminus S'$.*

(b) *For every vertex $a \in NR(C,Y) \cup \{u\}$, if a is isolated or semi-isolated in $G \setminus S$, then a is isolated or semi-isolated in $G \setminus S'$.*

Proof. **(a)** Consider a vertex x in $R(C,Y)$ which lies in a main component of $G \setminus S'$. Then, x lies on a T-path in $G \setminus S'$ and hence there are 2 vertex disjoint paths from x to T in $G \setminus S'$, which implies that there are 2 vertex disjoint paths from x to T in $G \setminus (Y \setminus \{u\})$. However, Y is an x-T separator in G by definition and hence there cannot be 2 vertex disjoint paths from x to T in $G \setminus (Y \setminus \{u\})$, a contradiction. Therefore, we conclude that every vertex in $R(C,Y)$ is either isolated or semi-isolated in $G \setminus S'$.

(b) Consider a vertex $a \in NR(C,Y) \cup \{u\}$ such that a is isolated or semi-isolated in $G \setminus S$, but lies in a main component of $G \setminus S'$. Since a does not have 2 vertex disjoint paths to T in $G \setminus S$, it must be the case that at least one of the two vertex disjoint paths from a to T in $G \setminus S'$ intersects the set $S \setminus S'$. This implies that there is a vertex $z \in S \setminus S'$ which has 2 vertex disjoint paths to T in $G \setminus S'$. However, this implies that $z \in X \setminus Y$, which is contained in $R(C,Y)$ by definition and we have already established that every vertex in $R(C,Y)$ is isolated or semi-isolated in $G \setminus S'$ (see **(a)**), which is a contradiction. This completes the proof of the claim. $\square$

We now compare the number of isolated and semi-isolated vertices in $G \setminus S$ against those in $G \setminus S'$. The number of isolated and semi-isolated vertices from the set $NR(C,Y) \cup \{u\}$ in $G \setminus S'$ is at least that in $G \setminus S$ (Claim 6 (b)). Similarly, the number of isolated and semi-isolated vertices from the set $R(C,Y) \setminus X$ in $G \setminus S'$ is also at least that in $G \setminus S$ (Claim 6 (a)). Therefore, it remains to compare the number of isolated and semi-isolated vertices in $G \setminus S$ which lie in $Y \setminus (X \cup \{u\})$ and the number of isolated and semi-isolated vertices in $G \setminus S'$ which lie in $X \setminus Y$. Again, by Claim 6 (a), the number of isolated or semi-isolated vertices in $G \setminus S'$ which lie in $X \setminus Y$ is exactly $|X \setminus Y|$. The number of isolated or semi-isolated vertices in $G \setminus S$ which lie in $Y \setminus (X \cup \{u\})$ is at most $|Y \setminus X| - 1$ since u is disjoint from $X \cap Y$. Therefore, the number of isolated or semi-isolated vertices in $G \setminus S'$ is at least $|X \setminus Y| - (|Y \setminus X| - 1)$ more than that in $G \setminus S$. Since by our assumption, $|Y| = |X|$, we have that that $|X \setminus Y| = |Y \setminus X|$. Therefore, we have a solution S' with strictly more isolated or semi-isolated vertices, which contradicts our choice of S. This completes the proof of the lemma. ☐

Lemma 7.3.15. *Every vertex $v \in V$ is contained in at most 4^{k+1} important components. Furthermore, all important components in G can be enumerated in time $\mathcal{O}^*(4^k)$.*

Proof. Consider an important component J. Then, $N(J)$ is an important v-T separator for any $v \in J$ (Lemma 3.2.17(1)). Since there are at most 4^{k+1} important v-T separators of size $k+1$, v can occur in at most 4^{k+1} important components. The set of important components can be computed by computing the set of important v-T separators for every vertex v in the graph. By Lemma 3.2.18, this requires time $\mathcal{O}^*(4^k)$. ☐

Lemma 7.3.16. *Let $G = (V, E)$ be a graph and T be a vertex set such that every vertex in G lies on a T-path and every T-path in G has the same parity. Then G is a bipartite graph.*

Proof. Suppose that this is not the case and let C be an odd cycle in G. Consider a vertex w in the cycle which lies on a t_1-t_2 path P where $t_1, t_2 \in T$. Clearly t_1 and t_2 cannot both lie on C, since it implies the presence of an odd T-path. Hence, at least one of the vertices, say t_1, is disjoint from C. Now, when traversing P from t_1 to t_2, let u be the first vertex of P which is in C and let v be the last vertex of P which is in C. Note that v and t_2 need not be distinct. Since P intersects w, it intersects C and hence the vertex u exists. Now, let P_1 be the subpath of P from t_1 to u and P_2 be the subpath of

P from v to t_2. Suppose that u and v are distinct and let P_o and P_e be the 2 subpaths of C between u and v where P_o is odd and P_e is even. Then, the two paths $P_1 + P_o + P_2$ and $P_1 + P_e + P_2$ are T-paths with different parities, which is a contradiction. Hence, it must be the case that $u = v$ and that any T-path intersecting w intersects C only in w. Now, let w' be a vertex in C distinct from w and let P' be a t_3-t_4 path on which w' lies, where $t_3, t_4 \in T$. Note that t_3 and t_4 need not be distinct from t_1 or t_2. Let P'_1 be the subpath of P' from t_3 to w' and let P'_2 be the subpath of P' from w' to t_4. By our earlier arguments, it must be the case that P'_1 and P'_2 intersect C only in the vertex w'. Now, if the paths P_1, P_2, P'_1 and P'_2 are all pairwise disjoint, then we have a T-path passing containing w which intersects C in at least 2 vertices, a contradiction to our earlier assumption that any T-path containing w does not intersect C in any other vertex. Hence, we consider the case when P_1 or P_2 intersects P. We assume without loss of generality that P'_1 intersects P. While traversing P'_1 from w' to t_3, let x be the first vertex of P on P'_1 and suppose that $x \in P_1$. Let the subpath of P'_1 from w' to x be P''_1 and the subpath of P_1 from t_1 to x be $\tilde{P}_1$. Then, the paths $\tilde{P}_1 + P''_1$ and P_2 are disjoint paths from distinct vertices of T to distinct vertices of C. This is a contradiction to our assumption that any T-path intersects C in at most 1 vertex. This completes the proof of the lemma. □

Observation 7.3.17. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let S be a solution for this instance. The main components of $G \setminus S$ are bipartite and any odd cycle in $G \setminus S$ has at most 1 vertex in a main component.*

Proof. Fix a main component in $G \setminus S$ and suppose that this component contains an odd cycle. This implies that the main component contains at least 3 vertices and therefore, there are at least 2 terminals in this main component. By Observation 7.2.2, the terminals in this main component are either all even terminals or all odd terminals. Therefore, the

T-paths which lie in this main component are all even or all odd. By Lemma 7.3.16, the main component is bipartite.

Suppose that there is an odd cycle in $G \setminus S$ which has two vertices in a main component. Since we have already shown that the main components are bipartite, this cycle must intersect a semi-isolated component as well. This implies that there is a semi-isolated vertex v which has 2 vertex disjoint paths to 2 distinct vertices in the adjacent main component, which is a contradiction to Lemma 7.3.6. □

Lemma 7.3.18. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC *and let S be a solution for this instance of the kind described in Lemma 7.3.14 and suppose that $G \setminus S$ is non-bipartite. Then, there is an important component C such that C is disjoint from S and $G[C \cup N(C)]$ is non-bipartite.*

Proof. Consider an odd cycle in $G \setminus S$. By Observation 7.3.17, at most one vertex of this cycle lies in a main component. Therefore, there is an isolated or semi-isolated component C such that this odd cycle lies in the graph $G[C \cup N(C)]$. However, by Lemma 7.3.14, every isolated and semi-isolated component in $G \setminus S$ is an important component and therefore, C is an important component disjoint from S such that $G[C \cup N(C)]$ is non-bipartite. This completes the proof of the lemma. □

We refer to any set which intersects all odd cycles in a graph as an *oct* of the graph. We now describe the procedure to obtain an instance of PMWC with a solution that is also an oct for the input graph. Formally, we have the following lemma.

Lemma 7.3.19. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* PMWC*. If $(G, T = T_e \cup T_o, k)$ is a* YES *instance, then there is a solution for this instance which is also an oct of G or there is a set Z of $2^{\mathcal{O}(k)}$ vertices which can be computed in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ such that Z intersects some solution for this instance.*

Proof. Suppose that $(G, T = T_e \cup T_o, k)$ is a YES instance and there is no solution for this instance which is also an oct of G. Let S be a solution for this instance such that every isolated or semi-isolated component is an important component. By Lemma 7.3.14, we know that such a solution exists. By our assumption, there is an odd cycle in $G \setminus S$.

Let $\mathcal{H}$ be the set of all important components in G. Let $\mathcal{J}$ be the set of all components $H \in \mathcal{H}$ such that $G[H \cup N(H)]$ is non-bipartite. Let $\mathcal{M}$ be an arbitrary subset of $\mathcal{J}$ of size $4^{k+1}k + 1$. If $|\mathcal{J}| < 4^{k+1}k + 1$, then set $\mathcal{M} = \mathcal{J}$. Let $M_1, \ldots, M_\ell$ be the components in $\mathcal{M}$. We now define the set Z as follows. Initially set $Z = \bigcup_{i=1}^{\ell} N(M_i)$. For each M_i, select two vertices u_i and v_i in an arbitrary odd cycle in $G[M_i \cup N(M_i)]$. For each $1 \leq i \leq \ell$, invoke the algorithms of Lemma 7.3.10 (Algorithm 7.3.1) and Lemma 7.3.12 (Algorithm 7.3.2) on the vertices u_i and v_i and add the vertices returned by each of these invocations to Z. Finally, return Z. We claim that Z intersects some solution for this instance. Suppose that this is not the case.

We first show that for some $M \in \mathcal{M}$, any odd cycle in $G[M \cup N(M)]$ is also present in $G \setminus S$.

We first consider the case when $|\mathcal{M}| = 4^{k+1}k + 1$. Since at most $4^{k+1}k$ important components can intersect S, there is an $M \in \mathcal{M}$ such that M is disjoint from S. Furthermore, by definition $N(M) \subseteq Z$ and by our assumption, Z is disjoint from S. Therefore, any odd cycle in the graph $G[M \cup N(M)]$ is an odd cycle in $G \setminus S$.

We now consider the case when $|\mathcal{M}| \leq 4^{k+1}k$. Recall that in this case, $\mathcal{M} = \mathcal{J}$. By Lemma 7.3.18, there is at least one important component M such that $G[M \cup N(M)]$ is non-bipartite. Since, every important component in G with the property that $G[C \cup N(C)]$ is non-bipartite is present in $\mathcal{J}$, M is also present in $\mathcal{M}$. Since by definition $N(M) \subseteq Z$ and by our assumption, Z is disjoint from S, we have that $N(M)$ is disjoint from S. Therefore, any odd cycle in the graph $G[M \cup N(M)]$ is an odd cycle in $G \setminus S$.

Without loss of generality, we assume that $M = M_1 \in \mathcal{M}$.

Recall that since we have assumed that Z is disjoint from S and Z contains $N(M_1)$ by definition, S is also disjoint from $N(M_1)$. Consider the vertices u_1 and v_1 selected by the algorithm from an odd cycle in $G[M \cup N(M)]$. By Observation 7.3.17, at least one of these two vertices is either isolated or semi-isolated in $G \setminus S$ and by Lemma 7.3.10 and Lemma 7.3.12, the sets returned by Algorithm 7.3.1 or Algorithm 7.3.2 indeed intersect some solution for the given instance. However, this leads to a contradiction since Z contains the vertices returned by these two invocations. Therefore, we conclude that Z intersects some solution for the given instance. It remains to prove the bound on the size of Z and the time required to compute Z.

The size of Z is bounded by $|\mathcal{M}|((k+1) + 2 \cdot 2^{\mathcal{O}(k)}) = 2^{\mathcal{O}(k)}$. The time required to compute Z is bounded by the time required to compute the set of important components in G and the time required to run Algorithm 7.3.1 and Algorithm 7.3.2 $2|\mathcal{M}|$ times. Therefore, Z can be computed in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$. This completes the proof of the lemma. ☐

Lemma 7.3.20. *Given an instance $(G, T = T_e \cup T_o, k)$ of* PMWC, *there is an algorithm which runs in time $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$ and returns ℓ instances $\{(G_i, T_i, k_i)\}_{1 \le i \le \ell}$ of* PMWC, *where $\ell = 2^{\mathcal{O}(k^2)}$ such that*

1. *If $(G, T = T_e \cup T_o, k)$ is a* YES *instance, then for some $1 \le i \le \ell$, the instance (G_i, T_i, k_i) is a* YES *instance of* PMWC *and has a solution which is also an oct of G_i.*

2. *If $(G, T = T_e \cup T_o, k)$ is a* NO *instance, then all the returned instances are* NO *instances of* PMWC.

Proof. The algorithm simply applies Lemma 7.3.19, constructs the set Z and branches on each vertex in this set. There is also an additional branch in which the current instance is returned. The correctness of this algorithm follows from Lemma 7.3.19. We now prove the stated bound on running time and the number of returned instances.

Since $|Z| = 2^{\mathcal{O}(k)}$, at every step, the algorithm branches in $2^{\mathcal{O}(k)}$ ways. Out of these branches, one is a leaf (where the current instance is returned) and in every other branch, k decreases by 1 since we are adding a vertex to the solution. Hence, the number of leaves in the recursion tree is bounded by $2^{\mathcal{O}(k^2)}$. The time taken at each step is the time required to compute Z, which is $\mathcal{O}^*(2^{\mathcal{O}(k)})$ by Lemma 7.3.19. Therefore, the algorithm runs in time $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$ and returns $2^{\mathcal{O}(k^2)}$ instances. This completes the proof of the lemma. $\square$

We apply the Lemma 7.3.20 each of the instances returned by the algorithm of Lemma 7.2.4 and therefore, we may assume that from this point on, we are dealing with the following problem.

BIPARTITION PARITY MULTIWAY CUT (BPMWC) **Parameter:** k
Input: An instance $(G, T = T_e \cup T_o, k)$ of PMWC where $|T_e| \leq 6k$, with the guarantee that if this instance is a YES instance, then there is a solution which is also an oct of G.
Question: Is there a pmwc of size at most k disjoint from T?

Before we move towards applying the important separator template to solve this problem, we perform one final preprocessing on the instance.

Lemma 7.3.21. *There is an algorithm that, given an instance $I = (G, T = T_e \cup T_o, k)$ of* BPMWC*, runs in time $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$ and returns $2^{k \log k}$ instances of* BPMWC *$\{(G_i, T = T_e \cup T_o, k)\}_{1 \leq i \leq \ell}$ where $\ell = 2^{k \log k}$ such that*

1. *If* $(G, T = T_e \cup T_o, k)$ *is a* YES *instance of* BPMWC *then there is an* $1 \leq i \leq \ell$ *such that* $I_i = (G_i, T = T_e \cup T_o, k)$ *is a* YES *instance of* BPMWC *and has an oct solution* S_i *such that if an even terminal is non-adjacent to the rest of the terminals, then it occurs by itself in a connected component of* $G_i \setminus S_i$.

2. *If* $(G, T = T_e \cup T_o, k)$ *is a* NO *instance* $I_i = (G_i, T = T_e \cup T_o, k)$ *is a* NO *instance of* BPMWC *for every* $1 \leq i \leq \ell$.

Proof. Let $\mathcal{P}$ be the set of all partitions of the set T_e with the property that no set in any partition contains more than 2 vertices of T_e. Since $T_e \leq 6k$, there are $2^{\mathcal{O}(k \log k)}$ such partitions. Let $\mathcal{P} = \{P_1, \ldots, P_\ell\}$. We construct ℓ instances of BPMWC each corresponding to a partition in $\mathcal{P}$ as follows. Consider a partition $P_i = (X_1, \ldots, X_r)$ where each X_j contains at most 2 vertices of T_e. Let G_i be the graph obtained from G by adding an edge between every pair of even terminals which occur in the same set in the partition P_i. Finally, the instances $\{(G_i, T = T_e \cup T_o, k)\}_{1 \leq i \leq \ell}$ are returned. The bound on the running time of the algorithm is clear.

1. Suppose that I is a YES instance and let S be a solution for this instance such that S is also an oct for G. Let $P = (X_1, \ldots, X_r)$ be the partition of T_e among the connected components of $G \setminus S$, that is, the vertices in each X_i occur in the same connected component of $G \setminus S$ and the vertices of any X_i and X_j occur in different components of $G \setminus S$ for $i \neq j$. Without loss of generality, let P_1 be this partition. We claim that the instance $I' = (G_1, T = T_e \cup T_o, k)$ is a YES instance of BPMWC. In order to prove this, we show that S is also a solution for the instance I' and S is an oct for the graph G_1.

If S were not a solution for the instance, then there is a T-path P of forbidden parity in $G_1 \setminus S$. Since S is a solution for $G \setminus S$, it must be the case that P contains an edge which is in G_1 but not in G. Therefore, P contains an edge between two even terminals t_1 and t_2. However, by assumption, t_1 and t_2 occur in the same component of $G \setminus S$ and there is no other terminal in this component. Since we only added edges between vertices in the same component of $G \setminus S$, t_1 and t_2 appear in the same connected component of $G_1 \setminus S$ and no other terminals occur in this component. Therefore, it must be the case that P is a path between t_1 and t_2. Since we have already established that P contains an edge between t_1 and t_2, P itself must be just the edge (t_1, t_2). But by assumption, P is a path of even parity, which contradicts our conclusion that P has length 1. Clearly, any even terminal in G_1 which is non-adjacent to the rest of the terminals occurs by itself in a connected component of $G \setminus S$ and hence occurs by itself in a connected component of $G_1 \setminus S$.

2. In the converse direction, since adding edges to the graph G cannot create a smaller pmwc, if I were a NO instance then $(G_i, T = T_e \cup T_o, k)$ will be a NO instance for every $1 \leq i \leq \ell$.

□

We assume that the instance of BPMWC we are given is one returned by the algorithm of Lemma 7.3.21. Furthermore, observe that in order to prove Theorem 7.1.1, it suffices to prove the following lemma.

Lemma 7.3.22. BPMWC *can be solved in time* $\mathcal{O}^*(2^{\mathcal{O}(k^3)})$.

In the rest of the chapter, we describe how to apply the important separator template to prove this lemma.

7.4 Phase 1 of the important separator template

The following lemma gives an algorithm for the case when the set of even terminals in the given instance is empty.

Lemma 7.4.1. *Let* $(G, T = T_e \cup T_o, k)$ *be an instance of* BPMWC *where* T_e *is empty. Then, we can solve this problem in time* $\mathcal{O}^*(2.3146^k)$.

Proof. The proof is by a parameter preserving reduction to the variable version of ALMOST 2-SAT, called the ALMOST 2 SAT(V) problem, which can then be solved in $\mathcal{O}^*(2.3146^k)$ time (Corollary 6.5.2). The reduction is as follows. For every vertex u of the graph, we have a variable x_u. The variable x_u is intended to represent the side of the (fixed) bipartition of $G \setminus S$ which contains u. The 2-CNF formula is constructed as follows. For every edge (t, u) where $t \in T$, add a clause (u). For every edge (u, v) in the graph, add two clauses $(x_u \vee x_v)$ and $(\bar{x}_u \vee \bar{x}_v)$ to the 2-CNF formula F. This completes the construction of F. We remark that both these clauses will be satisfied if and only if x_u and x_v are assigned different values. In addition, we also remark that the subformula F_P of F induced by the clauses corresponding to the edges of some odd T-path P is unsatisfiable, that is, we cannot find a satisfying assignment for it unless we delete at least 1 variable from this subformula. We claim that if $(G, T = T_e \cup T_o, k)$ is a YES instance of BPMWC, then (F, k) is a YES instance of ALMOST 2 SAT(V) and if (F, k) is a YES instance of ALMOST 2 SAT(V), then (G, T, k) is also a YES instance of BPMWC.

Suppose (G, T, k) has a solution S such that $G \setminus S$ is bipartite. Let S_v be the set of the variables corresponding to the vertices in S. We claim that the formula $F' = F \setminus S_v$ is satisfiable. Consider the following assignment for F'. Fix a bipartition for $G \setminus S$ such that all the vertices in T_o occur in the same partition. Such a partition is possible since for each connected component of $G \setminus S$, the vertices in T_o occur in the same partition of any bipartition. For every vertex u which lies in the same partition as T_o in the fixed bipartition of $G \setminus S$, we assign $x_u = 1$ and for all other vertices, we assign the corresponding variable the value 0. This assignment clearly satisfies F'.

Conversely, consider a solution S_v for the instance (F, k), let $F' = F \setminus S_v$, and let S be the set of vertices corresponding to the variables in S_v. If a terminal occurs in S, then we replace it with its neighboring vertex (recall that every odd terminal is adjacent to a single vertex). We claim that S is a solution for the given instance of BPMWC. Suppose that this is not the case, and consider an odd T-path P in the graph $G \setminus S$ and let F_P be the subformula of F induced by the clauses corresponding to the edges in P. Since none of the vertices intersecting the path P have been deleted, it must be the case that none of the variables corresponding to the vertices along this path are in S_v. But this implies that F_P remains as a subformula of F' and since F_P is not satisfiable, F' is also not satisfiable, which is a contradiction. □

Lemma 7.4.2. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* BPMWC *where T_o is empty and $|T_e| = 2$. Then, we can solve this problem in time $\mathcal{O}^*(2.3146^k)$.*

Proof. Let $T_e = \{t_1, t_2\}$. Let G' be the graph obtained from G by subdividing the edges incident on t_1, let $T'_e = \emptyset$ and let $T'_o = T_e$. It is easy to see that $(G, T = T_e \cup T_o, k)$ is a YES instance of BPMWC if and only if $(G', T' = T'_e \cup T'_o, k)$ is a YES instance of BPMWC. Therefore, by Lemma 7.4.1, the problem can be solved in time $\mathcal{O}^*(2.3146^k)$. □

Finally, the following lemma is a straightforward consequence of Lemma 7.3.21.

Lemma 7.4.3. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* BPMWC *where $|T_e| \geq 3$ or T_e and T_o are both non-empty. Let $T_1 \subseteq T_e$ such that either T_1 consists of a single even* *terminal non-adjacent to the rest of the terminals or it consists of a pair of adjacent even terminals. Then, there is an oct solution for the given instance containing a minimal T_1-$(T \setminus T_1)$ separator.*

We have therefore, shown that either the problem can be solved straightaway in FPT time by applying Lemma 7.4.1 or Lemma 7.4.2 or there is a solution which contains a minimal T_1-$(T \setminus T_1)$ separator. This completes Phase 1 of the important separator template.

7.5 Phases 2 and 3

In this section, we describe the second and third phases of the template.

7.5.1 Tight separator sequences and a generalization of important separators

In this subsection we define the notion of a tight separator sequence and use it in the context of BPMWC to define a generalization of important separators with the properties we require. This serves as the **dominating set** described in the important separator template.

Definition 7.5.1. *Let $G = (V, E)$ be a graph and let $X, Y \subseteq V$ be disjoint vertex sets. We define an* **important X-Y separator of order** i, *S^i to be the unique smallest important X-S^{i-1} separator in G, where $S^0 = Y$.*

Definition 7.5.2. *Let $G = (V, E)$ be a graph and let $X, Y \subseteq V$. Let $l \geq 1$ be such that there is no important X-Y separator of order $l + 1$. We define a* **tight X-Y separator sequence** *$\mathcal{I}$ to be a set $\mathcal{I} = \{S^i | 1 \leq i \leq l\}$, where S^i is an important X-Y separator of order i. (see Fig. 7.3).*

```
Input  : $(G, T_1, T_2, k)$
Output: Tight sequence of $T_1$-$T_2$ separators if there is a $T_1$-$T_2$ separator of size at
        most $k$ and NO otherwise
1 Apply Lemma 3.2.15 to compute the unique smallest important $T_1$-$T_2$ separator $Y$
2 $\mathcal{I} = Y$
3 if $|Y| > k$ then return NO.
4 while there is an important $T_1$-$Y$ separator $X$ of size at most $|Y|$ do
5     $\mathcal{I} = \mathcal{I} \cup \{X\}$
6     $Y = X$
7 end
8 return $\mathcal{I}$
```

Algorithm 7.5.1: Algorithm COMPUTE-TIGHT-SEQUENCE to compute the Tight Sequence of T_1-T_2 separators.

Observation 7.5.3. *Let $G = (V, E)$ be a graph and let $X, Y \subseteq V$. Given two X-Y separators S_1 and S_2, we say that $S_1 \preceq S_2$ if S_2 covers S_1 with respect to X. Then, $(\mathcal{I}, \preceq)$ forms a total order where $\mathcal{I}$ is a tight X-Y separator sequence.*

Observation 7.5.3 is the reason we refer to the set $\mathcal{I}$ as a *sequence*.

Lemma 7.5.4. *Let $G = (V, E)$ be a graph and let $X, Y \subseteq V$. A tight X-Y separator sequence is unique and can be computed in polynomial time.*

Proof. We know that for any two disjoint vertex subsets A and B, there is a unique smallest important A-B separator (by Lemma 3.2.13) and it can be computed in polynomial time. Therefore, for every i, an important X-Y separator of order i is unique and can be computed in polynomial time. The algorithm to compute the tight X-Y separator sequence is described in Algorithm 7.5.1. □

Lemma 7.5.5. *Let $G = (V, E)$ be a graph, let $X, Y \subseteq V$ and let $\mathcal{I}$ be the tight X-Y separator sequence. Let P_1 and P_2 be two separators in $\mathcal{I}$ such that $P_1 \preceq P_2$ and there is no P_3 in $\mathcal{I}$ such that $P_1 \preceq P_3 \preceq P_2$. Then, the size of a minimum X-Y separator which lies in the set $NR(X, P_1) \cap R(X, P_2)$ is at least $|P_1| + 1$.*

Proof. If this were not the case, then there is a an X-Y separator S of size $|P_1|$ which lies in the set $NR(X, P_1) \cap R(X, P_2)$. Since P_2 is a successor of P_1 in $\mathcal{I}$, it must be the case that $S \notin \mathcal{I}$. Let i be such that $P_2 = S^{i-1}$ and $P_1 = S^i$, that is, P_1 is the unique smallest important X-P_2 separator. However, since $P_1 \preceq S \preceq P_2$ and S lies in the set $NR(X, P_1) \cap R(X, P_2)$ it implies that S is an X-P_2 separator dominating P_1, which contradicts the fact that P_1 is an important X-P_2 separator. $\square$

The key consequence of the definition of the tight separator sequence is that it defines a natural partition of the graph into slices with *small boundaries*. Using this, we may restrict our search to local parts of the graph, in which case finding separators with certain properties becomes easier. We will now describe how this concept is applied in the context the BPMWC problem.

Definition 7.5.6. *For a graph $G = (V, E)$, we refer to a smallest odd cycle transversal in the graph G as a* **minimum oct** *and denote the size of this set by $oct(G)$.*

Definition 7.5.7. *Let $G = (V, E)$ be a graph, let $X, Y \subseteq V$. Let S be a minimal X-Y separator such that $oct(G[R(X, S)]) = \ell$. We say that a minimal X-Y separator S'* ***well dominates*** *S (with respect to X) if S' dominates S with respect to X and $oct(G[R(X, S')]) \leq \ell$.*

Note that any X-Y separator well dominates itself. The above definition is motivated by the following observation.

Observation 7.5.8. *Let $(G, T = T_e \cup T_o, k)$ be an instance of* BPMWC *and let $T_1 \subseteq T_e$ such that $T_1 = \{t\}$ where t is an even terminal non-adjacent to any other terminal or $T_1 = \{t_1, t_2\}$ where t_1 and t_2 are adjacent in G. Let S be a solution for this instance such that $G \setminus S$ is a bipartite graph. Let S' be a minimal part of S which separates T_1 from $T \setminus T_1$ in G and let $S'' = S \cap (R(T_1, S'))$. Then, S'' is an oct for $G[R(T_1, S')]$.*

Furthermore, if $|T_1| = 2$, then any oct for $G[R(T_1, S')]$ intersects all even T_1-paths in $G[R(T_1, S')]$.

Lemma 7.5.9. *Let $I = (G, T = T_e \cup T_o, k)$ be an instance of* BPMWC*, let S be an oct solution for this instance, and T_1 be a subset of T_e such that $|T_1| \leq 2$, and S is a T_1-T_2 separator where $T_2 = T \setminus T_1$. Furthermore, suppose that if $|T_1| = 2$, then the vertices in T_1 are adjacent. Let $\hat{S}$ be a minimal part of S separating T_1 and T_2. Let $\hat{S}_1$ be a T_1-T_2 separator which well dominates $\hat{S}$. Then, there is also an oct solution for the instance which contains $\hat{S}_1$.*

Proof. Let $\hat{K} \subseteq S$ be a minimum oct for the graph $G[R(T_1, \hat{S})]$ and let $\hat{K}_1$ be a minimum oct for the graph $G[R(T_1, \hat{S}_1)]$. Consider the set $S' = (S \setminus (\hat{S} \cup \hat{K})) \cup (\hat{S}_1 \cup \hat{K}_1)$. We claim that S' is also a solution for the given instance. It is clear that $|S'| \leq |S|$. Hence, it remains to show that S' is indeed a pmwc for the given instance and an oct of G.

Suppose that S' is not a pmwc for the given instance. Then, there are two terminals t_i and t_j such that there is a path P of forbidden parity between t_i and t_j in the graph $G \setminus S'$. Since S is a solution, this path intersects $S \setminus S' = \hat{S} \setminus \hat{S}_1$. That is, there is a vertex $v \in \hat{S} \setminus \hat{S}_1$ such that the path P intersects v. Suppose $t_i \in T_2$ or $t_j \in T_2$. Then, the presence of P in $G \setminus S'$ implies the presence of a path from v to T_2 in $G \setminus S'$. Since $\hat{S}_1 \subseteq S'$, there is a path from v to T_2 in $G \setminus \hat{S}_1$. However, since $\hat{S}_1$ dominates $\hat{S}$, there is no path from $\hat{S} \setminus \hat{S}_1$ to T_2 in the graph $G \setminus \hat{S}_1$ (by Lemma 3.2.9), which contradicts that there is a path from $v \in \hat{S} \setminus \hat{S}_1$ to T_2 in the graph $G \setminus \hat{S}_1$. Hence, we conclude that $t_i, t_j \in T_1$.

Since $|T_1| \leq 2$, $T_1 = \{t_i, t_j\}$. Since $T_1 \subseteq T_e$, the path P is an even t_i-t_j path. However, (t_i, t_j) is an edge (by our assumption). Therefore, by Observation 7.5.8, $\hat{K}_1$ intersects every even T_1 path in $G[R(T_1, \hat{S}_1)]$ and since $\hat{K}_1 \subseteq S'$, S' intersects every even T_1-path, a contradiction. Hence, we conclude that S' is a pmwc for the given instance. The same argument also shows that S' is an oct for G. This completes the proof of the lemma. $\square$

The above lemma describes the **dominating set** for the set of minimal T_1-T_2 separators and the following lemma shows that we can compute a bounded set of vertices intersecting a solution in FPT time.

Lemma 7.5.10. *Let* $(G, T = T_e \cup T_o, k)$ *be an instance of* BPMWC *with a solution* S*. Let* T_1 *be a subset of* T_e *such that* $|T_1| \leq 2$*, and* S *is a* T_1-T_2 *separator where* $T_2 = T \setminus T_1$*. Furthermore, if* $|T_1| = 2$*, then the vertices in* T_1 *are adjacent. Then, there is an algorithm which runs in time* $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$ *and returns a set of* $2^{\mathcal{O}(k^2)}$ *vertices which intersects some oct solution for the given instance.*

Proof. Let X be a minimal part of S separating T_1 from T_2. For a given set of *candidate* vertices, the algorithm computes (if possible) a subset of the candidate set which intersects a T_1-T_2 separator well dominating X. Initially, and also when the candidate set is not explicitly given, we allow this set to be the entire vertex set of the graph, and as we prune our search, we will redefine the candidate set accordingly. We first fix a hypothetical minimum oct for the graph $G[R(T_1, X)]$, say $K \subseteq S$ and guess the size of this set, say ℓ. Formally, given a tuple $(G, Z, T_1, T_2, k, \ell)$, the algorithm returns a set of vertices $\mathcal{R}$, such that for any T_1-T_2 separator X of size at most k with an oct of size ℓ in the graph $G[R(T_1, X)]$, there is a T_1-T_2 separator which well dominates X and has a non-empty intersection with $\mathcal{R}$. The algorithm is invoked on the input $(G, V \setminus (T_1 \cup T_2), T_1, T_2, k', \ell')$ for every $1 \leq k' \leq k$ and $1 \leq \ell' \leq k$.

Description of algorithm. We first check if there is a T_1-T_2 separator of size at most k contained in the given subset Z (see Algorithm 7.5.2). If not, then we return NO.

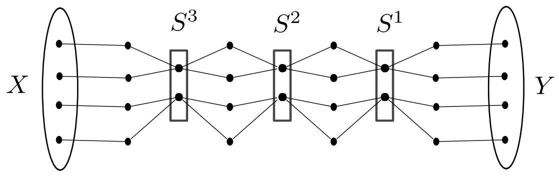

Figure 7.3: Illustration of a tight separator sequence

If there is no path from T_1 to T_2 in G, then we return $\emptyset$. Otherwise, we compute the tight T_1-T_2 separator sequence $\mathcal{I}$, comprising only the vertices in Z. We call a T_1-T_2 separator S' *good* if the size of the minimum oct in the graph $G[R(T_1, S')]$ is at most l and we call it *bad* otherwise. The following observation plays a crucial role in allowing us to ignore (potentially) large parts of the graph during our search.

Observation 7.5.11. *If a T_1-T_2 separator S' is* good, *then all T_1-T_2 separators covered by S' are also* good *and if S' is* bad, *then all T_1-T_2 separators which cover S' are* bad.

For each T_1-T_2 separator $Y \in \mathcal{I}$, we now determine whether Y is good or bad. For this we only need check if there is an oct of size at most ℓ in the graph $G[R(T_1, Y)]$ and hence this requires time $\mathcal{O}^*(2.3146^{\ell})$ (by Corollary 6.5.5).

Let P_1 be the maximal element of $\mathcal{I}$ which is good and let P_2 be the minimal element of $\mathcal{I}$, which is bad. That is, P_1 is good and every separator in $\mathcal{I} \setminus \{P_1\}$ which covers P_1 is bad, P_2 is bad and every separator in $\mathcal{I} \setminus \{P_2\}$ covered by P_2 is good. If all the separators in $\mathcal{I}$ are good, then P_2 is defined as T_2 and if all separators in $\mathcal{I}$ are bad, then P_1 is defined as T_1. We now move on to the description of the rest of the algorithm.

– We add the vertices in $P_1 \setminus T_1$ and $P_2 \setminus T_2$ into the set $\mathcal{R}$.

– We set $Z' = Z \cap (R(T_1, P_2) \cap NR(T_1, P_1))$ and add the vertices returned by the invocation **CWDS**$(G, Z', T_1, T_2, k, \ell)$ to $\mathcal{R}$.

```
    Input  : $(G, Z, T_1, T_2, k, \ell)$
    Output: A subset of $Z$ which, for every $T_1$-$T_2$ separator $X$, intersects a $T_1$-$T_2$
            separator well dominating $X$ or NO if no such separator exists
 1  if $k < 0$ then return NO
 2  Check if there is a $T_1$-$T_2$ separator of size at most $k$ contained in $Z$
 3  if there is no such separator then return NO
 4  if there is no $T_1$-$T_2$ path in $G$ then return $\emptyset$
 5  $\mathcal{I}$ =TIGHT-SEPARATOR-SEQUENCE$(G, T_1, T_2, k, \ell)$
 6  for each $Y \in \mathcal{I}$ do
 7      $O_Y \leftarrow oct(G[R(T_1, Y)])$
 8  end
 9  $P_1 \leftarrow$ maximal separator with $O_Y \leq \ell$
10  $P_2 \leftarrow$ minimal separator with $O_Y > \ell$
11  $\mathcal{R} \leftarrow (P_1 \setminus T_1) \cup (P_2 \setminus T_2)$
12  $Z' = Z \cap (R(T_1, P_2) \cap NR(T_1, P_1))$
13  $\mathcal{R} \leftarrow \mathcal{R} \cup$ CWDS$(G, Z', T_1, T_2, k, \ell)$
14  for each ordered 4-partition $J = (P_1^{nr}, P_1^e, \tilde{K}, P_1^o)$ of $P_1$ do
15      $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1, l' \leq \ell}$ CWDS$(G_J, Z \cap G_J, T_1, P_1^{nr}, k', \ell')$
16  end
17  for each ordered 4-partition $J = (P_2^{nr}, P_2^e, \tilde{K}, P_2^o)$ of $P_2$ do
18      $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1, \ell' \leq \ell}$CWDS$(G_J, Z \cap G_J, T_1, P_2^{nr}, k', \ell')$
19  end
20  return $R$
```

Algorithm 7.5.2: Algorithm COMPUTE-WELL-DOMINATING-SET (**CWDS**)

–For each ordered 4-partition of P_1, $J = (P_1^{nr}, P_1^e, \tilde{K}, P_1^o)$ we construct a graph G_J as follows. Initially, we set $G_J = G[R(T_1, P_1) \cup (P_1 \setminus \tilde{K})]$. For every pair in $P_1^e \times P_1^o$, we add an edge between the vertices and for every pair in $\binom{P_1^e}{2}$ and $\binom{P_1^o}{2}$ we add a subdivided edge between the vertices. This completes the construction of G_J.

Now, for each G_J, for each $1 \leq k' \leq k-1$ and for each $0 \leq \ell' \leq \ell$, we recurse on the instance $(G_J, Z \cap G_J, T_1, P_1^{nr}, k', \ell')$ and add the vertices in the sets returned, to $\mathcal{R}$.

–Similarly, we do the same for every ordered 4-partition of P_2.

– Finally, we return the set $\mathcal{R}$.

This completes the description of the algorithm. We now prove the correctness of the algorithm.

Correctness. For each tuple $I = (G, Z, T_1, T_2, k, \ell)$ on which the algorithm is invoked, we define a measure $\mu(I) = 2k - \lambda$ where k is the upper bound on the size of the T_1-T_2 separator we are searching for in I, and λ is the size of the smallest such separator. We prove the correctness of the algorithm by induction on the measure $\mu(I)$.

In the base case, if $\lambda > k$, then algorithm returns NO, which is clearly correct. Similarly, if $\lambda = 0$, then there is no path between T_1 and T_2 in the graph and hence the algorithm simply returns $\emptyset$ as the separator, which is also correct. This completes the correctness of the base cases and we now assume that the algorithm is correct on all instances I with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$.

1. Suppose P_1 covers the separator X or $P_1 = X$. In this case, the algorithm is correct on the instance I since P_1 itself is a T_1-T_2 separator well dominating X and the set returned by **CWDS**(I) contains P_1.

2. The separator X covers P_1, and X is covered by P_2 . Let $\tilde{S}_1$ be the intersection of X with P_1 and $\tilde{S}_2$ be the intersection of X with P_2. Suppose that $\tilde{S}_1 \cup \tilde{S}_2$ is non-empty. But $\tilde{S}_1 \cup \tilde{S}_2$ is contained in $P_1 \cup P_2$ and is disjoint from $T_1 \cup T_2$. Therefore, since the set returned by **CWDS**(I) contains $(P_1 \setminus T_1) \cup (P_2 \setminus T_2)$, the algorithm is correct on the instance I.

Now, consider the case when both $\tilde{S}_1$ and $\tilde{S}_2$ are empty. Let I' be the instance on which the algorithm is invoked in Step 13. By Lemma 7.5.5, any T_1-T_2 separator which lies in the set $NR(T_1, P_1) \cap R(T_1, P_2)$ has size at least $|P_1| + 1$. Hence, $\mu(I') \leq \mu - 1$ and by the induction hypothesis, **CWDS**(I') returns a set intersecting a T_1-T_2 separator of size at most k which well dominates X, which proves the correctness of the algorithm on the instance I as well.

3. The separator X is incomparable with P_1 . Let $\tilde{S}_1$ be the intersection of X with P_1, P_1^r be the intersection of P_1 with $R_G(T_1, X)$, $\tilde{K} = K \cap P_1$ and $P_1^{nr} = P_1 \setminus (P_1^r \cup \tilde{K})$. Also, let X^r be the intersection of X with $R_G(T_1, P_1)$ and let X^{nr} be the rest of X. Since X is incomparable with P_1, by Observation 3.2.8, P_1^r, X^r, P_1^{nr} and X^{nr} are non empty. Let P_1^e and P_1^o be a bipartition of $P_1^r \setminus \tilde{K}$ such that the vertices in P_1^e and P_1^o appear in different partitions of some bipartition of $G \setminus S$ ($G \setminus S$ is bipartite since S is an oct for G).

Since the set returned by **CWDS**(I) contains $P_1 \setminus T_1$, if $\tilde{S}_1$ is non-empty then the algorithm is correct on I. Now, consider the case when $\tilde{S}_1$ is empty. Consider the graph G_J corresponding to the partition $(P_1^{nr}, P_1^e, \tilde{K}, P_1^o)$. Also, let $K^r = K \cap R(T_1, P_1)$ and let $\ell' = |K^r|$. Let I' be the instance $(G_J, Z \cap G_J, T_1, P_1^{nr}, |K^r|, \ell')$. Recall that the set returned by the invocation **CWDS**(I) contains the vertices in the set returned by the invocation **CWDS**(I') (Line 15 of the algorithm). We now prove the following claim which will be used along with the induction hypothesis to prove the correctness of the algorithm on I.

Claim 7. ***(a)*** *The set X^r is a T_1-P_1^{nr} separator in the graph G_J such that the subgraph $G_J[R_{G_J}(T_1, X^r)]$ has an oct of size ℓ'.*

(b) *Let X' be a T_1-P_1^{nr} separator which well dominates X^r in the graph G_J. Then, the* *set $X' \cup X^{nr}$ well dominates X in the graph G.*

Proof. **(a)** In order to prove this statement, it is sufficient to show that K^r is an oct for the graph $G' = G_J[R_{G_J}(T_1, X^r)]$. If this were not the case, then there is an odd cycle C in the graph $G' \setminus K^r$. Replace any edge (respectively subdivided edge) of C which is present in G_J, but not present in G, with a corresponding path of the same parity in G (recall that this corresponding path is also present in $G \setminus S$ to obtain a closed walk W. Observe that W only contains vertices which are disjoint from S. Therefore, it must be the case that W is an odd walk in the graph $G \setminus S$. However, this contradicts our assumption that S is an oct for G.

(b) Since X' well dominates X^r in G_J, $X'' = X' \cup X^{nr}$ is a T_1-T_2 separator dominating X since it separates T_1 from P_1^{nr} and T_2 from P_1^r and $|X'| \leq |X^r|$ implies that $|X''| \leq |X|$. Therefore, it is sufficient to show that the graph $G' = G[R(T_1, X'')]$ has an oct of size at most ℓ. We claim that the set $K'' = K' \cup (K \setminus K^r)$ is indeed such an oct for G'. It is clear that the size of K'' is at most that of K. Since $|K| = \ell$, $|K''| \leq \ell$. Therefore it remains to show that K'' is indeed an oct for G'.

If this were not the case, then there is an odd cycle C in the graph $G' \setminus K''$. Clearly, C cannot intersect P_1^{nr} since X'' separates T_1 from P_1^{nr} by definition. Suppose that C lies in the set $P_1^r \cup NR(T_1, P_1)$. Then, C also lies in the set $P_1^r \cup NR(T_1, P_1)$ in the graph $G[R(T_1, X)]$. But K is an oct for $G[R(T_1, X)]$ and since K^r lies in $R(T_1, P_1)$ it must be the case that $K \setminus K^r$ intersects C. However, $K \setminus K^r$ is also contained in K'', which implies that K'' intersects C, a contradiction. Therefore we may assume that C intersects the set $R(T_1, P_1)$.

Now, consider any subpath of C lying in $P_1^r \cup NR(T_1, P_1)$ with only the endpoints in P_1^r and the remaining vertices in $NR(T_1, P_1)$. If this path is odd, then there is an edge between these two vertices in G_J and if this path is even, then there is a subdivided edge between these two vertices in G_J (by definition of G_J). Therefore, we can replace all such sub-paths of C with the corresponding edge or subdivided edge in G_J to obtain an odd cycle C' in the graph $G_J[R_{G_J}(T_1, X')]$ disjoint from the set K', which is a contradiction. This completes the proof of the claim. $\square$

Now, let X' be a T_1-P_1^{nr} separator which well dominates X^r in the graph G'. Then, due to Claim 7, $X' \cup X^{nr}$ is a T_1-T_2 separator which well dominates X in the graph G. Therefore, a set intersecting a T_1-P_1^{nr} separator which well dominates X^r in the graph G' also intersects a T_1-T_2 separator which well dominates X in the graph G. Therefore, if the algorithm is correct on I', then it is also correct on I and hence it only remains to prove that the algorithm is correct on I'. Note that to prove that the algorithm is correct on I', it is sufficient to prove that $\mu(I') < \mu(I) = \mu$ since the correctness then follows from the induction hypothesis.

By Menger's theorem, since P_1 is a minimum size T_1-T_2 separator, we know that there are P_1^{nr} vertex disjoint paths from T_1 to P_1^{nr}, which is also a lower bound on the size of the smallest T_1-P_1^{nr} separator in G_J. Now, $\mu(I) = 2(|X^r| + |X^{nr}|) - (|P_1^{nr}| + |P_1^r|)$ and $\mu(I') = 2|X^r| - |P_1^{nr}|$, which implies that $\mu(I') = \mu(I) - (2|X^{nr}| - |P_1^r|)$. Since $|X^{nr}| \geq |P_1^r|$, we have that $\mu(I) - \mu(I') \geq |X^{nr}|$. Since X^{nr} is non empty, we conclude that $\mu(I') < \mu(I)$, which completes the proof of correctness of this case.

4. The separator X is incomparable with P_2. This correctness of this case is analogous to the correctness of the previous case.

We note that the separator X is a good separator by definition and therefore cannot cover P_2 due to Observation 7.5.11. Therefore the case that X covers P_2 need not be taken into consideration and the cases we have considered are exhaustive. This completes the proof of correctness of the algorithm.

Running time. We show by induction on $\mu(I)$ that the number of vertices in the set returned by **CWDS**(I), denoted by $N(\mu(I))$, is bounded by $2^{6\mu(I)^2}$. In each of the base cases of the algorithm, we either return a single vertex or say NO, and hence the bound clearly holds. We assume that the claimed bound is true for all instances with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$. Moreover, $k > 1$.

1. The number of vertices added in Step 11 is bounded by $2k$.

2. The number of vertices added in Step 13 is bounded by $2^{6(\mu-1)^2}$ by the induction hypothesis.

3. Consider the vertices added in Step 15. There are at most 4^k ordered 4-partitions of P_1, k choices for ℓ', and k choices for k'. For each of these choices, we return a set of size at most $N(\mu - 1)$, which, by the induction hypothesis is at most $2^{6(\mu-1)^2}$. Hence, the number of vertices added in this step is bounded by $4^k \cdot k^2 \cdot 2^{6(\mu-1)^2}$.

4. Similarly, the number of vertices added in Step 18 is bounded by $4^k \cdot k^2 \cdot 2^{6(\mu-1)^2}$.

Using the fact that $k \leq \mu$ and $k \geq 1$, we note that the total number of vertices returned by **CWDS**(I) is at most $2^{6(\mu)^2}$, which yields the required bound. The number of leaves of the recursion tree is clearly bounded by the size of the set returned by the algorithm, which is $2^{\mathcal{O}(k^2)}$, which is also a bound on the number of internal nodes of the recursion tree. Since the algorithm spends $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time at each node of the

Phase 1.

(a) If $T_e = \emptyset$ or $|T_e| = 2$ and $T_o = \emptyset$, then the problem can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ (Lemma 7.4.2 and Lemma 7.4.1).

(b) If neither of the above cases are true, then there is a set $T_1 \subseteq T_e$ such that $|T_1| \leq 2$ and the solution is a T_1-T_2 separator (Lemma 7.4.3).

Phase (2 + 3). There is a set $\mathcal{R}$ of $2^{\mathcal{O}(k^2)}$ vertices which can be computed in time $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$ such that there is an oct solution for the given instance intersecting $\mathcal{R}$ (Lemma 7.5.10).

Phase 4. If either case of Phase 1 applies, then solve the problem by applying Lemma 7.4.1 or Lemma 7.4.2. Otherwise, run the algorithm of Lemma 7.5.10 and branch on the vertices returned by this algorithm.

Figure 7.4: Summary of the phases of the important separator template

search tree, the total time taken by the algorithm on the given input, for a fixed k and ℓ is $\mathcal{O}^*(2^{\mathcal{O}(k)})$. Since we invoke the Algorithm **CWDS** for every $1 \leq k' \leq k$ and $1 \leq \ell' \leq k$, the total number of vertices returned is $2^{\mathcal{O}(k^2)}$. This completes the proof of the lemma. □

The phases are summarized in Figure 7.4. It is easy to see that the running time of the algorithm for BPMWC is $\mathcal{O}^*(2^{\mathcal{O}(k^3)})$. This completes the proof of Theorem 7.1.1. □

Part IV

Parameterized Approximations

8

Important Separators and Approximation

8.1 FPT-approximation

Definition 8.1.1. (*see, e.g.,*([28])) *An* FPT-approximation algorithm *with ratio* ρ *for a minimization (maximization) problem* P *is an* FPT *algorithm that, given an instance* x *of* P *and a positive integer* l*, either determines that there is no solution of size at most* l *(respectively at least* l*) or computes a solution of cost at most* $k\rho(k)$ *(respectively at least* $k/\rho(k)$*) where* k *is the parameter. We say that a problem is* FPT*-approximable if it has an* FPT*-approximation algorithm for some function* ρ*.*

Note that the approximation ratio ρ is a function of the parameter and not the input size. In this chapter, we describe an important separator approach to obtain $poly(k)$ factor FPT-approximation algorithms for graph separation problems.

8.2 The Template

The Important Separator Approximation Template

1. Show that either there is a solution which contains an X-Y separator of bounded size for two sets X and Y or there is a branching or reduction which simplifies the problem, or the problem is exactly solvable in FPT time.
2. Show that in FPT time, we can either conclude correctly that there is no X-Y separator of the required size or compute an X-Y separator S such that the size of S is bounded by a function of k and there is a solution for the given instance intersecting the set $R(X, S) \cup S$.
3. Show that the instance resulting from deleting the separator computed in the previous phase has a strictly smaller solution.
4. Combine the phases to get a greedy approximation algorithm.

In the following section, we illustrate this approach by describing a polynomial time approximation algorithm for the MULTIWAY CUT problem and in the next chapter, we give an FPT-approximation algorithm using this template.

8.3 The template applied to MULTIWAY CUT

The best known polynomial time approximation for the MULTIWAY CUT problem has an approximation factor of $(2 - \frac{2}{l})$ where l is the number of terminals in the input instance [37]. However, this algorithm requires the solution of a relaxation of the Integer Linear Program for the MULTIWAY CUT problem. In this section, we demonstrate how to apply the template presented in the previous section to obtain a simple greedy factor-OPT polynomial time approximation algorithm for MULTIWAY CUT where the super-linear dependence of the running time is restricted to be polynomially bounded by only OPT, with the dependence on the input size remaining linear.

Recall that the problem we consider is defined as follows.

MULTIWAY CUT	**Parameter:** k
Input: A graph G, vertex subset T, positive integer k	
Question: Is there a set S of at most k vertices which intersects every T-path in G?	

We also refer to the set S in the above definition as a *mwc* for the given instance.

Phase 1. For any terminal $t_1 \in T$, if t_1 is disconnected from the rest of the terminals in T, then we can clearly simplify the instance further by removing the component containing t_1 from the graph. Otherwise, any solution must contain a minimal t_1-$T \setminus t_1$ separator.

Phase 2.

Lemma 8.3.1. *Let $(G = (V, E), T, k)$ be an instance of* MULTIWAY CUT *and let $t_1 \in T$ be a terminal such that t_1 lies in the same connected component of G as some vertex in $T \setminus t_1$. Then, there is an algorithm that runs in time $\mathcal{O}(k^3(m+n))$ and either concludes that there is no t_1-$T \setminus t_1$ separator of size at most k or computes a t_1-$T \setminus t_1$ separator S of size at most k such that any solution for the given instance solution intersects the set $R(t_1, S) \cup S$ where $n = |V|$ and $m = |E|$.*

Proof. Consider an important t_1-$T \setminus t_1$ separator S of size at most k such that there is no important t_1-$T \setminus t_1$ separator S' of size at most k which covers S. Recall that S' covers S if $R(t_1, S') \supset R(t_1, S)$. Let X be a solution for the given instance of MULTIWAY CUT. We claim that X intersects the set $R(t_1, S) \cup S$. Since we know that X is a t_1-$T \setminus t_1$ separator and there is a path in G from t_1 to a vertex in $T \setminus t_1$, there is a non-empty set $X' \subseteq X$ such that X' is a minimal t_1-$T \setminus t_1$ separator of size at most k. By our choice of S, it cannot be the case that X' covers S. Therefore, X' intersects the set $R(t_1, S) \cup S$. Observe that S can be chosen as any t_1-$T \setminus t_1$ separator of size at most k which is computed at a leaf of the search tree of algorithm which computes all important t_1-$T \setminus t_1$ separators. Such a leaf can be reached in the claimed time by the following algorithm. We first compute the unique minimum important t_1-$T \setminus t_1$ separator S and if it has size less than k, check if there is a vertex $v \in S$ such that there is an $R(t_1, S) \cup \{v\}$-$T \setminus t_1$ separator of size at most k and if so, select such a separator as S and continue. Since each time we chose the unique minimum important separator, we cannot find more than k such separators and the time taken in moving from one separator to the next is bounded by the time taken for k applications of the algorithm of Lemma 3.2.15, which is $\mathcal{O}(k(m + n))$, the running time follows. This completes the proof of the lemma. ☐

Phase 3.

Lemma 8.3.2. *Let (G, T, k) be a* YES *instance of* MULTIWAY CUT *and let $t_1 \in T$ be a terminal such that t_1 lies in the same connected component of G as some vertex in $T \setminus t_1$. Let S be a t_1-$T \setminus t_1$ separator S of size at most k such that any solution for the given instance intersects the set $R(t_1, S) \cup S$. Then, $(G \setminus S, T, k - 1)$ is a* YES *instance of* MULTIWAY CUT.

Proof. Let X be a solution for the instance (G, T, k). Let $X' = X \cap (R(t_1, S) \cup S)$ and let $X'' = X \setminus X'$. Since X' is non-empty, $|X''| \leq k - 1$. We claim that X'' is a solution for the instance $(G \setminus S, T, k)$. If this were not the case, then there is a t_i-t_j path P in $G \setminus (S \cup X'')$ where $t_i, t_j \in T$. Since S intersects all paths in G from t_1 to $T \setminus t_1$, $t_i, t_j \neq t_1$. Since X is a solution for the instance (G, T, k), the path P intersects a vertex in $X \setminus (X'' \cup S)$, which implies that P intersects a path in $X' \setminus S$. However, observe that $X' \setminus S \subseteq R_G(t_1, S)$, which implies that P intersects a vertex in $R_G(t_1, S)$ and a vertex in T, but is disjoint from S, which is a contradiction since S is a t_1-$T \setminus t_1$ separator. This completes the proof of the lemma.

□

Phase 4. We combine the 3 phases and describe our approximation algorithm for MULTIWAY CUT as follows. Given an instance (G, T, k), check if there is a component of G containing at least 2 terminals in T. If there is no such component, then the empty set is the solution. On the other hand, if there is such a component then select an arbitrary terminal t_1 from such a component and apply Lemma 8.3.1 to compute a t_1-$T \setminus t_1$ separator S of size at most k. Then, return the set obtained by the union of S and the result of the recursion on the instance $(G \setminus S, T, k-1)$.

Analysis of the algorithm. The fact that this algorithm returns a set intersecting every T-path follows from Lemma 8.3.1 and Lemma 8.3.2. The fact that this algorithm runs in time $\mathcal{O}(k^4(m+n))$ follows from the fact that the algorithm of Lemma 8.3.1 runs in time $\mathcal{O}(k^3(m+n))$ and there are at most k applications of this algorithm.

Furthermore, since the number of recursions is bounded by k and each step of the recursion adds a set of size at most k to the mwc we construct, the set returned is bounded by k^2. This completes the analysis of the algorithm and therefore we have the following lemma.

Lemma 8.3.3. *There is an algorithm that, given an instance $(G = (V, E), T, k)$ of* MULTIWAY CUT, *runs in time $\mathcal{O}(k^4(m+n))$ and either returns a mwc of size at most k^2 or concludes correctly that there is no mwc of size at most k where $n = |V|$ and $m = |E|$.*

We conclude this chapter with the next lemma which is a straightforward consequence of Lemma 8.3.3.

Lemma 8.3.4. *There is an algorithm for the optimization version of* MULTIWAY CUT *that, on an instance* $(G = (V, E), T)$ *runs in time* $\mathcal{O}(OPT^5(m+n))$ *and returns a mwc of size at most* OPT^2*, where **OPT** is the size of the smallest mwc for the given instance,* $n = |V|$ *and* $m = |E|$.

9

Backdoors to q-Horn

9.1 Introduction

SATISFIABILITY (SAT) is a well-known fundamental problem in Computer Science. SAT and its various variants like its optimization version (finding the maximum number of clauses that can be satisfied by a truth assignment) and its generalizations (CONSTRAINT SATISFACTION PROBLEM (CSP)) frequently occur in several practical applications [5, 71, 97, 61]. Many hard combinatorial problems can be encoded as SAT instances in the broad sense mentioned above, including problems that arise in hardware and software verification, Artificial Intelligence (AI) planning and scheduling, OR resource allocation, etc. However, the problem is known to be NP-hard and thus we cannot hope to find a solution to this ubiquitous problem in polynomial time. In spite of this, over the last two decades, SAT-solvers have become amazingly successful in solving formulas with hundreds of thousands of variables that encode problems arising from various application areas, see, e.g., [43]. But theoretical performance guarantees are far from explaining this empirically observed efficiency. In fact, there is an enormous gap between theory and practice.

The discrepancy between theory and practice can be explained by the presence of a certain “hidden structure” in real-world problem instances. One such “hidden structure” in real-world instances of SAT is the presence of *small backdoor sets*. Williams et al. [99] introduced the notion of backdoor sets with respect to a tractable base class to explain the surprisingly good performance of state of the art SAT solvers. There are three variants of backdoor sets with respect to a particular base class $\mathcal{C}$: strong $\mathcal{C}$-backdoor sets, where for each truth assignment to the backdoor variables, the reduced formula belongs to $\mathcal{C}$; deletion $\mathcal{C}$-backdoor sets (or variable deletion control set [22]), where deleting all backdoor variables and their negations from the formula moves the formula into the base class $\mathcal{C}$; and weak backdoor sets where, for at least one truth assignment to the backdoor variables, the reduced formula belongs to $\mathcal{C}$ and is satisfiable. Given a backdoor set of a formula with respect to a particular tractable base class $\mathcal{C}$, the satisfiability of the formula can be decided by guessing an assignment to the variables in the backdoor set and deciding the satisfiability of the reduced formula, which is guaranteed to be in $\mathcal{C}$, using a sub-solver for $\mathcal{C}$. This motivates the problem of computing the smallest backdoor set of a given formula with respect to a fixed base class.

The parameterized complexity of the problem of finding small backdoor sets was initiated by Nishimura et al. [86] who showed that for the base classes of Horn formulas and 2CNF formulas, the detection of strong backdoor sets is fixed-parameter tractable. Their algorithms exploit the fact that for these two base classes, strong and deletion backdoor sets coincide, and that deletion backdoor sets with respect to Horn and 2CNF can be characterized in terms of vertex covers and hitting sets of certain graphs and 3-uniform hypergraphs associated with the input formula, respectively. For base classes other than Horn and 2-CNF, strong backdoor sets can be much smaller than deletion backdoor sets, and their detection is more difficult. In particular, for the base classes of renamable Horn and q-Horn, there are formulas that have a strong backdoor set of size 1 but require an arbitrarily large deletion backdoor set. In fact, Razgon and O'Sullivan [90] showed that the detection of deletion backdoor sets with respect to the base class renamable Horn is fixed-parameter tractable although the detection of strong backdoor sets is W[2]-hard [39]. For more recent results, the reader is referred to a recent survey on the parameterized complexity of backdoor sets [39].

In this chapter, we consider as our base class a class of CNF formula called q-Horn, which was introduced by Boros, Crama and Hammer [8]. This class has favorable algorithmic properties: both recognition as well as deciding satisfiability of q-Horn formulas can be done in linear-time [8, 9]. Furthermore, this class properly contains the fundamental classes of Horn and 2CNF formulas [93], and the class of renamable (or disguised) Horn formulas [66, 3]:

$$\begin{tikzcd}
\text{Horn} \arrow[r, phantom, "\subsetneq"] & \text{renamable Horn} \arrow[dr, phantom, "\subsetneq"] & \\
& & \text{q-Horn} \\
& \text{2CNF} \arrow[ur, phantom, "\subsetneq"] &
\end{tikzcd}$$

The fact that this class is so large serves as an additional motivation for choosing it as our base class of interest. In this chapter, we study the problem of finding small backdoor sets with respect to the class of q-Horn formulas and obtain algorithmic as well as hardness results.

The main result of this chapter is the following theorem.

Theorem 9.1.1. *There is an algorithm that, given an instance (F, k) of* Deletion q-Horn Backdoor Set Detection*, runs in time $\mathcal{O}(6^k k\ell n)$ and either correctly concludes that F has no deletion* q-Horn *backdoor set of size at most k or returns a deletion* q-Horn *backdoor set of F of size at most $2k^2 + 2k$, where ℓ is the length of F and n is the number of variables in F.*

A consequence of this theorem is the fixed-parameter tractability of CNF-SAT (or SAT) with the size of the smallest q-Horn deletion backdoor set as the parameter, as we can use this algorithm to reduce the satisfiability problem of a CNF formula F of distance k from being q-Horn to testing the satisfiability of $4^{(k^2+k)}$ many q-Horn formulas.

Overview of our algorithm. At the highest level, our algorithm works by finding a bounded number of variables whose deletion results in an instance with an optimum solution strictly smaller than that of the original instance. By repeatedly computing such a set and deleting it, we obtain the approximate solution. The main technical part of the paper is the algorithm to compute the bounded set of variables with the required properties. For this, we utilize a characterization of q-Horn formulas in terms of their quadratic cover by Boros, Hammer, and Sun [9] which allows us to model the DELETION q-Horn BACKDOOR SET DETECTION problem as a graph separation problem in an auxiliary digraph related to the formula. Following this, we apply the approximation framework described in the previous chapter to compute in polynomial time a set of variables whose size is bounded by $2k$ such that (a) there is a minimal (though not necessarily optimum) solution containing these variables and (b) deletion of these variables results in a formula whose solution is strictly smaller than the solution for the formula we started with. A standout feature of our algorithm is that at its core, it reduces to computing flows in a directed graph whose size is linear in the input size. As a result, our algorithm is quite efficient not only with respect to the dependence of the running time on the parameter, but also with respect to the dependence on the input size, along with having only a small hidden constant factor in the asymptotic running time. Finally, towards the end of the chapter, we also provide parameterized complexity results regarding the detection of weak and strong backdoor sets with respect to the class q-Horn.

9.2 Preliminaries

Here, we introduce the basic terminology for backdoors and the class of q-Horn formulas. For further information on backdoors and other tractable base classes of SATISFIABILITY we refer the reader to [39].

Backdoors are defined with respect to a fixed class $\mathcal{C}$ of CNF formulas, the *base class* (or *target class*, or more figuratively, *island of tractability*). From a base class we require the following properties: (i) $\mathcal{C}$ can be recognized in polynomial time, (ii) the satisfiability of formulas in $\mathcal{C}$ can be decided in polynomial time, and (iii) $\mathcal{C}$ is closed under isomorphisms (i.e., if two formulas differ only in the names of their variables, then either both or none belong to $\mathcal{C}$). We say a class $\mathcal{C}$ of formulas is *clause-induced* if it is closed under subsets, , if $F \in \mathcal{C}$ implies $F' \in \mathcal{C}$ for each $F' \subseteq F$.

A *strong $\mathcal{C}$-backdoor set* of a CNF formula F is a set B of variables such that $F[\tau] \in \mathcal{C}$ for each $\tau \in 2^B$. A *weak $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F[\tau]$ is satisfiable and $F[\tau] \in \mathcal{C}$ holds for some $\tau \in 2^B$. A *deletion $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F \setminus B \in \mathcal{C}$.

If we know a strong $\mathcal{C}$-backdoor set of F of size k, we can reduce the satisfiability of F to the satisfiability of 2^k formulas in $\mathcal{C}$. Thus SAT becomes fixed-parameter tractable with k as the parameter. If we know a weak $\mathcal{C}$-backdoor set of F, then F is clearly satisfiable, and we can verify it by trying for each $\tau \in 2^k$ whether $F[\tau]$ is in $\mathcal{C}$ and satisfiable. If $\mathcal{C}$ is clause-induced, any deletion $\mathcal{C}$-backdoor set of F is a strong $\mathcal{C}$-backdoor set of F. For several base classes, deletion backdoor sets are of interest because they are easier to detect than strong backdoor sets. The challenging problem is to find a strong, weak, or deletion $\mathcal{C}$-backdoor set of size at most k if it exists. For each class $\mathcal{C}$ of CNF formulas, the various backdoor detection problems are defined as follows.

DELETION $\mathcal{C}$-BACKDOOR SET DETECTION **Parameter:** k

Input: A CNF formula F and a positive integer k

Question: Does F have a deletion $\mathcal{C}$-backdoor set of size at most k?

The problems STRONG $\mathcal{C}$-BACKDOOR SET DETECTION and WEAK $\mathcal{C}$-BACKDOOR SET DETECTION are defined similarly.

In this chapter, we are mainly interested in the class of q-Horn formulas [8, 9]. A CNF formula F is in this class if there is a *certifying function* $\beta : \text{var}(F) \cup \overline{\text{var}(F)} \rightarrow \{0, \frac{1}{2}, 1\}$ with $\beta(x) = 1 - \beta(\bar{x})$ for every $x \in \text{var}(F)$ such that $\sum_{l \in C} \beta(l) \leq 1$ for every clause C of F. Note that the class of q-Horn formulas satisfies our requirements (i)–(iii) on base classes for backdoor sets [8, 9].

In the following sense, strong q-Horn-backdoor sets are more general than deletion q-Horn-backdoor sets: For every positive integer n there is a formula F_n such that F_n has a strong q-Horn-backdoor set of size 1 but every deletion q-Horn-backdoor set of F has size at least n. To see this, take for instance $F = \bigcup_{1 \leq i \leq n} \{\{x_i, y_i, z_i, a\}, \{\bar{x}_i, \bar{y}_i, \bar{z}_i, \bar{a}\}\}$. Evidently, $\{a\}$ is a strong q-Horn backdoor set of F. However, every deletion q-Horn backdoor set of F must contain at least one variable x_i, y_i, or z_i for every $1 \leq i \leq n$.

9.3 Quadratic covers, implication graphs and separators

In this section, we give some definitions regarding quadratic covers, implication graphs and separators in implication graphs, which will be required for the description of our algorithm.

The following definition of the quadratic cover of a CNF formula was used to give a linear time algorithm to recognize q-Horn formulas in [9].

Definition 9.3.1. *Given a CNF formula F, the* **quadratic cover** *of F, is a 2-CNF formula denoted by F_2 and is defined as follows. Let $x_1, \ldots, x_n$ be the variables of F. For every clause C, we have $|C| - 1$ new variables $y_1^C, \ldots, y_{|C|-1}^C$. We order the literals in each clause according to their variables, that is a literal of x_i will occur before a literal of x_j if $i < j$. Let $l_1^C, \ldots, l_{|C|}^C$ be the literals of the clause C in this order. The quadratic cover is defined as*

$$F_2 = \bigcup_{C\in F} \bigcup_{1\le i\le |C|-1} \{\{l_i^C, y_i^C\}, \{\bar{y}_i^C, l_{i+1}^C\}\} \quad \cup \quad \bigcup_{C\in F} \bigcup_{1\le i\le |C|-2} \{\{\bar{y}_i^C, y_{i+1}^C\}\}.$$

Definition 9.3.2. *Given a CNF formula F, the* **implication graph** *of F_2 is denoted by D(F_2) and defined as follows. The vertex set of the graph is the set of literals of F_2 and for every clause $\{l_1, l_2\}$ in F_2, we have arcs $(\bar{l}_1, l_2)$ and $(\bar{l}_2, l_1)$. We refer to the vertices of the implication graph as literals since there is a one to one correspondence between the two. Given a set of variables $X \subseteq var(F)$, we define the graph D(F_2)$\setminus X$ as the graph obtained from D(F_2) by deleting lit(X).*

The following observation stems from the definition of implication graphs.

Observation 9.3.3. *Let F be a CNF formula of length ℓ.*

(a) *If there is a path from l_1 to l_2 in D(F_2), then there is also a path from $\bar{l}_2$ to $\bar{l}_1$ in D(F_2).*

(b) *The number of arcs in D(F_2) is $\mathcal{O}(\ell)$.*

(c) *Let $C = \{l_1, \ldots, l_r\}$ be a clause of F. Then, for any $1 \le i < j \le r$, D(F_2) contains a path from $\bar{l}_i$ to l_j and from $\bar{l}_j$ to l_i whose internal vertices are all disjoint from lit(F).*

(d) *Let $X \subseteq var(F)$ and $F' = F \setminus X$. Then, for any literal $l \in$ lit(F) $\setminus$ lit(X), there is a path from l to $\bar{l}$ in D(F_2') if and only if there is a path from l to $\bar{l}$ in D(F_2)$\setminus X$.*

Proof. **(a)** This follows from the fact that for every arc (l_i, l_j) in D(F_2), there is also an arc $(\bar{l}_j, \bar{l}_i)$.

(b) Observe that a clause $C \in F$ contributes at most $3|C|$ clauses to F_2. Hence, the number of clauses in F_2 is bounded by 3ℓ. Since each clause of F_2 contributes 2 arcs to D(F_2), the number of arcs in D(F_2) is bounded by 6ℓ.

(c) From the definition of the formula F_2 and the implication graph D(F_2), we know that D(F_2) contains a directed path $P_1 = y_1^C, y_2^C, \ldots, y_{r-1}^C$. Furthermore, there are arcs $(\bar{l}_i, y_i^C)$ and (y_{j-1}^C, l_j) in D(F_2). Along with the path P_1, these arcs imply that there is a path from $\bar{l}_i$ to l_j and by (a), a path from $\bar{l}_j$ to l_i. Clearly, in both these paths, the internal vertices are disjoint from lit(F).

(d) In order to prove the statement it is sufficient to show that for any two literals l_1 and l_2 such that var$(l_1) \neq$ var(l_2), there is a path from l_1 to l_2 in D(F_2') if and only if there is a path from l_1 to l_2 in D(F_2)$\backslash X$. Furthermore, in order to prove this statement, it suffices to show that this statement is true for all paths where the internal vertices are not in lit(F). The statement will then follow by induction on the number of literals of F along the paths.

Consider a path from l_1 to l_2 in D(F_2') where the internal vertices are all disjoint from lit(F). We claim that $\bar{l}_1$ and l_2 occur in a clause in F'. This is so since the definition of F_2' and D(F_2') implies that any literal not in lit(F) corresponds to a single clause of F' and has arcs only to or from other literals of this type which also correspond to this clause or to a literal in lit(F) which occurs positively in the same clause or from a literal in lit(F) which occurs negatively in the same clause. Therefore, $\bar{l}_1$ and l_2 also occur in a clause in F. This implies that there is a path from l_1 to l_2 in D(F_2) where the internal vertices are disjoint from lit(F) (by (c)). Therefore, removing lit(X) from D(F_2) does not affect this path.

Conversely, consider a path from l_1 to l_2 in D(F_2)$\setminus X$ such that the internal vertices are disjoint from lit(F). Since this is also a path in D(F_2), by our previous argument, it implies that $\bar{l}_1$ and l_2 occur in a clause in F and hence in F' and by (c), there is a path from l_1 to l_2 in D(F_2') where the internal vertices are disjoint from lit(F). $\square$

Definition 9.3.4. *Given a CNF formula F and a set L of literals of F, we denote by $N_F^+(L)$ the set of literals in lit$(F) \setminus L$ which can be reached from L in D(F_2) via a path whose internal vertices are disjoint from lit(F). We drop the subscript D(F_2) if the formula F is clear from the context.*

Definition 9.3.5. ([9]) *Given a CNF formula F, define a canonical function $\hat{\beta}$: lit$(F) \rightarrow \{0, \frac{1}{2}, 1\}$ as follows. Consider a topological ordering π of the strong components of D(F_2). For every literal $l \in$ lit(F) such that the strong component containing l appears before the one containing $\bar{l}$ in π, set $\hat{\beta}(l) = 1$ and for every literal l such that the strong component containing l also contains $\bar{l}$, set $\hat{\beta}(l) = \frac{1}{2}$.*

Lemma 9.3.6. ([9]) *A CNF formula F is* q-Horn *if and only if the function $\hat{\beta}$ defined above is a certifying function for F.*

Definition 9.3.7. *A clause C of a given CNF formula is called a* **violating clause** *if $\sum_{l\in C} \hat{\beta}(l) > 1$. Any three literals l_1, l_2, l_3 of a violating clause such that $\sum_{i=1}^{3} \hat{\beta}(l_i) > 1$ form a* **violating triple**.

Lemma 9.3.8. *Let F be a CNF formula of length ℓ and suppose that F is not a* q-Horn *formula. Any violating clause of F has a violating triple lying entirely inside a strong component of D(F_2) and we can compute such a violating triple in time $\mathcal{O}(\ell)$.*

Proof. Consider a violating clause $C = \{l_1, \dots, l_r\}$. We first show that for any l_i, $\hat{\beta}(l_i) \leq \frac{1}{2}$. Suppose this is not the case and there is an l_i such that $\hat{\beta}(l_i) = 1$. Since C is a violating clause, there is also a literal l_j such that $\hat{\beta}(l_j) > 0$. It follows that l_j and $\bar{l}_j$ either occur in the same strong component of D(F_2) or the strong component containing l_j occurs before that containing $\bar{l}_j$. However, we already know that l_i is in a strong component occurring before $\bar{l}_i$. But, there is an $\bar{l}_i$-l_j path and an $\bar{l}_j$-l_i path in D(F_2) (Observation 10.2.1(c)), implying that $\bar{l}_i$ is in a strong component occurring before the strong component containing l_i, a contradiction. Therefore for any l_i, $\hat{\beta}(l_i) \leq \frac{1}{2}$.

Since C is a violating clause, there must be at least three literals l_i, l_j, l_m such that $\hat{\beta}(l_i) = \hat{\beta}(l_j) = \hat{\beta}(l_m) = \frac{1}{2}$. By the definition of $\hat{\beta}$, each pair of complementary literals appear in the same strong component of D(F_2). However, there are paths from $\bar{l}_i$ to l_j and l_m and from $\bar{l}_j$ and $\bar{l}_m$ to l_i (Observation 10.2.1(c)). This implies that the literals l_i, l_j, l_m and their complements all appear in the same strong component of D(F_2). Observe that to compute the violating triple, it is sufficient to construct D(F_2) and compute a violating clause, which can be done in time $\mathcal{O}(\ell)$ [9]. $\square$

We now move on to some definitions on separators in implication graphs which will be required in the description of our algorithm.

Definition 9.3.9. *Let F be a CNF formula and $L \subseteq lit(F)$ be a consistent set of literals. We say that a set $J \subseteq lit(F)$ is an L-$\bar{L}$* **separator** *if J is disjoint from L and $\bar{L}$ and there is no path from L to $\bar{L}$ in the graph D(F_2)$\setminus J$. We say that J is a minimal L-$\bar{L}$ separator if no proper subset of J is an L-$\bar{L}$ separator.*

Definition 9.3.10. *Let F be a CNF formula, $L \subseteq lit(F)$ be a consistent set of literals and let X be a set of variables of F. We call X an L-$\bar{L}$* **variable separator** *if $lit(X)$ is an L-$\bar{L}$ separator. We call X a minimal L-$\bar{L}$ variable separator if no proper subset of X is an L-$\bar{L}$ variable separator. We drop the word* variable *if it is clear from the context that the set we are dealing with is a set of variables.*

Definition 9.3.11. *Let F be a CNF formula, $L \subseteq lit(F)$ be a consistent set of literals and X be an L-$\bar{L}$ variable separator. We denote by $R(L, X)$ the set of literals of F that can be reached from L via directed paths in D(F_2)$\setminus X$ and we denote by $\bar{R}(L, X)$ the set of literals of F which have a directed path to L in D(F_2)$\setminus X$.*

The following observation is implicit in the proof of Observation 10.2.1(d). We state it explicitly in the form in which it will be used as part of the description of our algorithm.

Observation 9.3.12. *Let F be a CNF formula, $L \subseteq lit(F)$ be a consistent set of literals and X be an L-$\bar{L}$ variable separator. Then, the sets $R(L, X)$ and $\bar{R}(\bar{L}, X)$ are also consistent and in fact complements of each other.*

9.4 Phase 1

Lemma 9.4.1. *Let (F, k) be an instance of* DELETION q-Horn BACKDOOR SET DETECTION. *Let (l_1, l_2, l_3) be a violating triple in a strong component of D(F_2) and X be an optimum solution for the given instance disjoint from $\{var(l_1), var(l_2), var(l_3)\}$. Then, for some $1 \leq i \leq 3$, X is an l_i-$\bar{l}_i$ separator in D(F_2).*

Proof. Let $\hat{\beta}'$ be the canonical certifying function for $F' = F \setminus X$ obtained from the graph D(F_2'). We claim that there is an $1 \leq i \leq 3$ such that $\hat{\beta}'(l_i) = 0$. This is true since F' contains a clause with all three literals l_1, l_2 and l_3 and it cannot be the case that any certifying function sets non zero values to all three. By definition of $\hat{\beta}'$, $\hat{\beta}'(l_i) = 0$ implies that there is no path from l_i to $\bar{l}_i$ in the graph D(F_2'). If X were not an l_i-$\bar{l}_i$ separator in D(F_2), then D(F_2') would also contain an l_i-$\bar{l}_i$ path (by Observation 10.2.1(d)), a contradiction. This completes the proof of the lemma. □

Lemma 9.3.8 combined with Lemma 9.4.1 allows us to compute in linear time, a set of three literals such that either one of the three corresponding variables is part of an optimum solution or for at least one of these literals, say l, there is a path from l to $\bar{l}$ in D(F_2) and there is an optimum solution which is an l-$\bar{l}$ variable separator in D(F_2).

9.5 Phase 2

Lemma 9.5.1. *Let F be a CNF formula of length ℓ, with n variables. Given a literal l of F and an integer k, there is an algorithm running in time $\mathcal{O}(k\ell n)$ which either correctly concludes that there is no k-sized l-$\bar{l}$ variable separator in D(F_2) or returns an l-$\bar{l}$ variable separator X' of size at most $4k$ such that $(X' \cup \text{var}(R(l, X'))) \cap X$ is non-empty where X is an l-$\bar{l}$ variable separator of size at most k.*

Proof. The algorithm computes an l-$\bar{l}$ variable separator X' which essentially maximizes the set of literals of D(F_2) reachable from l after removing X'. We will then show that such a separator indeed has the required properties.

The algorithm is called COMPUTE-SEPARATOR (Algorithm 9.5.1). If it returns No in Line 4, then D(F_2) has no l-$\bar{l}$ variable separator of size at most k. Let S be the minimal separator in D(F_2) which was computed in the penultimate iteration of the while loop. We claim that $X' = \text{var}(S)$ satisfies the conditions in the statement of the lemma. Clearly, it must be the case that for some choice of a literal l' in $\text{lit}(\text{var}(S)) \cap N_F^+(L)$, the next iteration of the loop could not find an $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l}'\}$ separator of size at most $2k$.

Suppose that $(X' \cup \text{var}(R(l, X'))) \cap X$ is empty. Recall that when the procedure stops, $L = R(l, X')$. Furthermore, if there is at least one path from l to $\bar{l}$ in D(F_2) then it must be the case that $\text{lit}(\text{var}(S)) \cap N_F^+(L)$ is non-empty. Since X is an l-$\bar{l}$ separator and disjoint from L, X is also an L-$\bar{L}$ separator. Since X is also disjoint from X', for any $l' \in \text{lit}(\text{var}(S)) \cap N^+(L)$, X intersects all paths from $L \cup \{l'\}$ to $\bar{L} \cup \{\bar{l}'\}$. Hence,

```
Input  : A tuple $(F, k, l)$ where $F$ is a CNF formula, $k$ a positive integer and $l$ a
         literal of $F$
Output: NO, provided that D($F_2$) has no $l$-$\bar{l}$ variable separator of size at most $k$,
         or an $l$-$\bar{l}$ variable separator $S$ of size at most $4k$ such that
         $(S \cup \text{var}(R(l, S)))$ has a non-empty intersection with some minimum
         deletion q-Horn backdoor set
1  if there is an $l$-$\bar{l}$ separator of size at most $2k$ in D($F_2$) then
2      $S \leftarrow$ such a separator
3  end
4  else return NO
5  $L \leftarrow R(l, \text{var}(S))$ // $L$ is consistent by Observation 10.2.5
6  while there is an $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l}'\}$ separator of size at most $2k$ where
   $l' \in (lit(var(S)) \cap N_F^+(L))$ is an arbitrarily chosen such literal do
7      $S \leftarrow$ such a separator
8      $L \leftarrow R(L, \text{var}(S))$
9  end
10 return var($S$)
```

Algorithm 9.5.1: Algorithm COMPUTE-SEPARATOR

lit(X) is a set of size at most $2k$ which intersects all $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l}'\}$ paths, which is a contradiction. Therefore, the set $(X' \cup \text{var}(R(l, X'))) \cap X$ is non-empty for any l-$\bar{l}$ variable separator X of size at most k.

To bound the running time, observe that in each iteration, we only need to test if there is an L-$\bar{L}$ separator of size at most $2k$. Hence, it suffices for us to run the Ford-Fulkerson algorithm [35] for at most $2k$ steps on the graph D(F_2) and the number of iterations is bounded by the number of variables in the formula since in each iteration, we add a literal to L. Since the number of arcs in D(F_2) is $\mathcal{O}(\ell)$ (Observation 10.2.1(b)), the claimed time bound follows. This completes the proof of the lemma. □

9.6 Phase 3

Lemma 9.6.1. *Let (F, k) be an instance of* DELETION q-Horn BACKDOOR SET DETECTION *and X be an optimum solution such that it is disjoint from var(l) and is an l-$\bar{l}$ separator for some literal $l \in$ lit(F). Consider a minimal l-$\bar{l}$ variable separator X'. Let X'' be the set of variables of X with a literal in $R(l, X')$. Then, the set $\tilde{X} = (X \setminus X'') \cup X'$ is also a solution for the given instance.*

Proof. Let $F' = F \setminus X$ and $\tilde{F} = F \setminus \tilde{X}$. If $\tilde{X}$ were not a solution, then there is a violating clause in $\tilde{F}$ and by Lemma 9.3.8, there is a violating triple (l_1, l_2, l_3) in a strong component of D($\tilde{F}_2$). This implies the presence of a closed walk in D($\tilde{F}_2$) containing all the literals of the violating triple and their complements (see the proof of Lemma 9.3.8). Since X was a solution, this closed walk could not have survived in D(F'_2) and hence must contain a literal of a variable in $X \setminus \tilde{X}$. Recall that the only variables of X that are not in $\tilde{X}$ are those in X''. Let p be a literal on this closed walk which corresponds to such a variable, that is var$(p) \in X''$. On the other hand, by definition, the literals of the variables in X'' can either reach $\bar{l}$ or be reached from l in D($\tilde{F}_2$), that is they must lie in $R(l, \tilde{X})$ or $\bar{R}(l, \tilde{X})$. Combining this path along with the closed walk and the fact that D($\tilde{F}_2$) is an implication graph implies the presence of a path from l to $\bar{l}$ in D($\tilde{F}_2$). However, by construction, $\tilde{X}$ is also an l-$\bar{l}$ separator in D(F_2). Observation 10.2.1(d) implies that this is a contradiction. This completes the proof of the lemma. □

Lemma 9.6.2. *Let (F, k) be an instance of* DELETION q-Horn BACKDOOR SET DETECTION *and let l be a literal of F disjoint from an optimum solution X and suppose that X is an l-$\bar{l}$ variable separator in D(F_2). Consider a minimal l-$\bar{l}$ variable separator in D(F_2), X', such that $(X' \cup var(R(l, X'))) \cap X$ is non-empty. Then the size of an optimum solution for the instance $F \setminus X'$ is at most $|X| - 1$.*

Phase 1. Lemma 9.3.8 combined with Lemma 9.4.1 allows us to compute in linear time, a set of three literals such that either one of the three corresponding variables is part of an optimum solution or for at least one of these literals, say l, there is a path from l to $\bar{l}$ in D(F_2) and there is an optimum solution which is an l-$\bar{l}$ variable separator in D(F_2).

Phase 2 . Lemma 9.5.1 gives an algorithm which either correctly concludes that there is no l-$\bar{l}$ variable separator of size at most k or returns an l-$\bar{l}$ variable separator of size at most $4k$.

Phase 3. Lemma 9.6.2 shows that deleting the separator returned by Lemma 9.5.1 results in an instance with a solution strictly smaller than that of the given instance.

Figure 9.1: Summary of the first 3 phases of the Important Separator Approximation Template

Proof. By Lemma 9.6.1, we know that the set $\hat{X} = (X \setminus X'') \cup X'$ is a deletion q-Horn backdoor set. Hence, $X \setminus X''$ is indeed a solution for the instance $F \setminus X'$. Since X'' is non-empty, the size of the optimum solution for $F \setminus X'$ is at most $|X| - 1$. This completes the proof of the lemma. □

This completes Phase 3 and we now ready to combine the phases to give an algorithm for DELETION q-Horn BACKDOOR SET DETECTION.

9.7 Phase 4

Description of the algorithm. The algorithm (Algorithm 9.7.1) checks if there is a violating triple and if so, computes one and in the first 3 branches, it adds the variable corresponding to each of the literals of the violating triple to the solution, deletes it from the formula and recurses on the resulting instance with a budget of $k - 1$. In the next 3 branches, it guesses the literal of the violating triple which is assigned 0 by a certifying

```
Input  : A CNF formula $F$ of length $\ell$ with $n$ variables, integer $k$
Output: Either no solution of size at most $k$ or a solution of size at most $2k^2 + 2k$
        for the instance $(F, k)$ of DELETION q-Horn BACKDOOR SET
        DETECTION
1  if $k < 0$ then return NO
2  check for a violating clause by computing D($F_2$) and a topological ordering of
   D($F_2$)
3  if there is no violating clause then return $\emptyset$
4  Compute a violating triple $(l_1, l_2, l_3)$
5  for $l = l_1, l_2, l_3$ do
6      $S_1 \leftarrow$ DELETION-QHORN-BDS($F \setminus \{var(l)\}, k - 1$)
7      if $S_1$ is not NO then return $S_1 \cup \{var(l)\}$
8  end
9  for $l = l_1, l_2, l_3$ do
10     $S \leftarrow$ COMPUTE-SEPARATOR($F, k, l$)
11     if $S$ is NO then return NO else
12         $S_1 \leftarrow$ DELETION-QHORN-BDS($F \setminus \{S\}, k - 1$)
13     end
14     if $S_1$ is not NO then return $S_1 \cup \{S\}$
15 end
16 return NO
```
Algorithm 9.7.1: Algorithm DELETION-QHORN-BDS for DELETION q-Horn BACKDOOR SET DETECTION

function of $F \setminus X$ where X is an optimum solution. We know that there must be at least one such literal (see the proof of Lemma 9.4.1). This implies that there is a path from l to $\bar{l}$ in D(F_2) and that X is an l-$\bar{l}$ separator. Finally, Lemma 9.5.1 is used to either conclude that there is no l-$\bar{l}$ variable separator of size at most k in which case the algorithm returns NO, or to compute an l-$\bar{l}$ variable separator of size at most $4k$ with the required properties. The variables in X' are added to our solution and deleted from the formula, and the algorithm recurses on the resulting instance with a budget of $k - 1$.

Analysis. Since Steps 2, 4, and 10 at any node of the search tree take time $\mathcal{O}(k\ell n)$ and we have a 6-way branching at each node of the search tree with the budget k dropping by 1 in each branch, the algorithm clearly runs in the claimed time bound. Therefore, it only remains for us to prove the correctness of the algorithm. Let X be an optimum solution for the given instance and let β be a certifying function for $F \setminus X$. We prove the correctness of the algorithm by induction on k.

In the base case, when $k = 0$, the algorithm is correct by Lemma 9.3.6. We assume as induction hypothesis that the algorithm is correct for all values of k up to some $k' - 1$ where $k' - 1 > 0$. We now consider the case when $k = k'$.

In Lines 5-8, we consider the case when X intersects the set $\{\text{var}(l_1), \text{var}(l_2), \text{var}(l_3)\}$ and branch accordingly. Applying the induction hypothesis, the size of any returned solution in a subsequent recursive call is at most $2(k-1)^2 + 2(k-1)$. Hence, the size of a solution returned here is bounded by $1 + 2(k-1)^2 + 2(k-1) \leq 2k^2 + 2k$.

In Lines 9-15, we consider the case when X is disjoint from the set of variables corresponding to l_1, l_2 and l_3. Since l_1, l_2, l_3 lie in the same clause and none of their corresponding variables are in X, by Lemma 9.4.1, X is an l_i-$\bar{l}_i$ separator for at least one of the literals l_i. Let us assume that this literal is l_1. In Line 10, we apply Lemma 9.5.1 to compute an l_1-$\bar{l}_1$ separator S of size at most $4k$ and add it to the solution we are constructing. By Lemma 9.6.2, we know that the size of an optimum solution for the instance $F \setminus S$ is at most $|X| - 1$. Hence, by the induction hypothesis, we obtain a solution of size at most $2(k-1)^2 + 2(k-1)$ from the subsequent recursive call and adding to it the set S of size at most $4k$ results in a solution of size at most $2k^2 + 2k$, which proves the correctness of the algorithm, completing the proof of Theorem 9.1.1.

In order to test the satisfiability of a given CNF formula F, it suffices to first compute a smallest deletion q-Horn backdoor set of F and for each assignment to this set, test the satisfiability of the reduced formula which is q-Horn. Since testing satisfiability of a q-Horn formula is linear time [8], Theorem 9.1.1 has the following corollary.

Corollary 9.7.1. *There is an algorithm that, given a formula F of length ℓ with n vari-*

ables, runs in time $4^{(k^2+k)}\ell n$ *and decides the satisfiability of* F*, where* k *is the size of the smallest deletion* q-Horn *backdoor set of* F*.*

9.8 Hardness

In this section, we show that there is no FPT algorithm for STRONG q-Horn-BACKDOOR SET DETECTION or WEAK q-Horn-BACKDOOR SET DETECTION unless FPT=W[2]. In order to show this, we begin from the following problem, which is well-known to be W[2]-complete [27].

HITTING SET **Parameter:** k

Input: A set E of elements, a family $\mathcal{S}$ of finite subsets of E, and an integer $k > 0$.

Question: Does $\mathcal{S}$ have a hitting set, i.e., a subset H of E such that $H \cap S \neq \emptyset$ for every $S \in \mathcal{S}$, of size at most k?

Theorem 9.8.1. STRONG q-Horn-BACKDOOR SET DETECTION *is* W[2]*-hard.*

Proof. We prove the theorem via a parameterized reduction from HITTING SET. Let $(E, \mathcal{S}, k)$ be an instance of HITTING SET. We construct a formula F that has a strong q-Horn-backdoor set of size at most k if and only if $\mathcal{S}$ has a hitting set of size at most k. The formula F has two clauses $P_S^i = S \cup \{x_i, y_i, z_i\}$ and $N_S^i = \bar{E} \cup \{\bar{x}_i, \bar{y}_i, \bar{z}_i\}$ for every $S \in \mathcal{S}$ and $1 \leq i \leq k+1$. Note that $\text{var}(F) = E \cup \{\, x_i, y_i, z_i : 1 \leq i \leq k+1 \,\}$. Furthermore, for any S and for any $1 \leq i \leq k+1$, the formula comprising the two clauses P_S^i and N_S^i is clearly not q-Horn.

Let H be a hitting set of $\mathcal{S}$ of size at most k. We show that $B = H$ is a strong q-Horn-backdoor set for F. Let $\tau : B \to \{0, 1\}$ be an assignment of the variables in B. If τ sets at least one variable in B to 0 then $F[\tau]$ contains only positive clauses and hence $F[\tau]$ is trivially q-Horn. On the other hand if τ sets all variables in B to 1 then because

H is a hitting set of $\mathcal{S}$ the formula $F[\tau]$ contains only negative clauses and hence $F[\tau]$ is again trivially q-Horn. Consequently, B is a strong q-Horn-backdoor set for F.

For the reverse direction suppose that $B \subseteq \text{var}(F)$ is a strong q-Horn-backdoor set for F of size at most k. We show that $H = B \cap E$ is a hitting set of $\mathcal{S}$ of size at most k. Suppose for a contradiction that H is not a hitting set for $\mathcal{S}$ and let $S \in \mathcal{S}$ be such that $H \cap S = \emptyset$. Furthermore, let τ be the assignment of the variables in B that sets all variables to 1. It follows that $F[\tau]$ contains two clauses P_S^i and N_S^i for some $1 \leq i \leq k+1$ and hence $F[\tau]$ is not q-Horn, a contradiction to the assumption that B is a strong q-Horn-backdoor set for F. $\square$

Theorem 9.8.2. WEAK q-Horn-BACKDOOR SET DETECTION *is* W[2]-*hard, even for* 3-*CNF formulas.*

Proof. We prove the theorem via a parameterized reduction from HITTING SET. Let $(E, \mathcal{S}, k)$ be an instance of HITTING SET. We construct a 3-CNF formula F that has a weak q-Horn-backdoor set of size at most k if and only if $\mathcal{S}$ has a hitting set of size at most k. For every $S \in \mathcal{S}$ with $S = \{s_1, \dots, s_{|S|}\}$, every $1 \leq i \leq |S|$, and every $1 \leq j \leq k+1$ the formula F contains the clauses $\{z_i^j(S), \bar{s}_i, \bar{z}_{i+1}^j(S)\}$, $\{\bar{z}_1^j(S), z_{|S|+1}^j(S)\}$, $\{\bar{z}_1^j(S), \bar{z}_{|S|+1}^j(S)\}$, $\{z_1^j(S), z_{|S|+1}^j(S)\}$, $\{z_{s+1}^j(S), a^j(S), b^j(S)\}$, and $\{\bar{a}^j(S), \bar{b}^j(S)\}$. Note that $\text{var}(F) = E \cup \{\, z_i^j(S) : S \in \mathcal{S} \text{ and } 1 \leq i \leq |S|+1 \text{ and } 1 \leq j \leq k+1 \,\} \cup \{\, a^j(S), b^j(S) : S \in \mathcal{S} \text{ and } 1 \leq j \leq k{+}1 \,\}$. Note furthermore that F is satisfiable by the assignment τ_{SAT} that sets the variables in $\{\, z_{|S|+1}^j(S), a^j(S) : S \in \mathcal{S} \text{ and } 1 \leq j \leq k{+}1 \,\}$ to 1 and all other variables to 0.

Let H be a hitting set of $\mathcal{S}$ of size at most k. We show that $B = H$ is a weak q-Horn-backdoor set for F. Let τ be the assignment of the variables in B that sets all variables to 0. Then $F[\tau]$ is satisfiable because τ assigns the same values to the variables in B as τ_{SAT}. To show that $F[\tau]$ is q-Horn we define the certifying function

$\beta : \text{lit}(F[\tau]) \to \{0, \frac{1}{2}, 1\}$ as follows. We set $\beta(e) = 1$ for every $e \in E \setminus H$ and for every $S \in \mathcal{S}$ with $S = \{s_1, \dots, s_{|S|}\}$, every $1 \leq j \leq k+1$, and some $1 \leq h \leq |S|$ such that $s_h \in S \cap H$, we set $\beta(a^j) = 1$, $\beta(b^j) = 0$, $\beta(z_i^j) = 1$ for every $1 \leq i \leq h$, and $\beta(z_i^j) = 0$ for every $h < i \leq |S| + 1$. It is straightforward to check that β certifies that F is q-Horn formula.

For the reverse direction suppose that $B \subseteq \text{var}(F)$ is a weak q-Horn-backdoor set for F of size at most k and let τ be an assignment for the variables in B witnessing this. We show that $H = B \cap E$ is a hitting set of $\mathcal{S}$ of size at most k. Suppose for a contradiction that H is not a hitting set for $\mathcal{S}$ and let $S \in \mathcal{S}$ be such that $H \cap S = \emptyset$. It follows that there is a $1 \leq j \leq k+1$ such that $F[\tau]$ contains the clauses $\{z_i^j(S), \bar{s}_i, \bar{z}_{i+1}^j(S)\}$, $\{\bar{z}_1^j(S), z_{|S|+1}^j(S)\}$, $\{\bar{z}_1^j(S), \bar{z}_{|S|+1}^j(S)\}$, $\{z_1^j(S), z_{|S|+1}^j(S)\}$, $\{z_{s+1}^j(S), a^j(S), b^j(S)\}$, and $\{\bar{a}^j(S), \bar{b}^j(S)\}$ for every $1 \leq i \leq |S|$. It is straightforward to verify that the subformula of $F[\tau]$ induced on the above clauses, and hence also $F[\tau]$, is not a q-Horn formula, contradicting our assumption that B is a weak q-Horn backdoor set for F. $\square$

It remains an open problem whether STRONG q-Horn-BACKDOOR SET DETECTION or WEAK q-Horn-BACKDOOR SET DETECTION are FPT-approximable. However we note that since the reductions used in the above theorems are parameter preserving, an FPT-approximation algorithm for either of these problems would imply the existence of an FPT-approximation algorithm for HITTING SET, which is an open problem [74].

9.9 Conclusion

In this chapter we have developed an FPT-approximation algorithm for the detection of deletion q-Horn-backdoor sets (Theorem 9.1.1). This renders CNF-SAT, parameterized by the *deletion distance* from the class of q-Horn-formulas (i.e., the size of a smallest q-Horn-backdoor set) fixed-parameter tractable (Corollary 9.7.1). Our result simultaneously generalizes the known fixed-parameter tractability results for CNF-SAT parameterized by the deletion distance from the class of renamable Horn formulas [90] and from the class of 2CNF formulas [86]. We would like to point out that our FPT-approximation algorithm is quite efficient, and its asymptotic running time does not include large hidden factors.

The deletion distance from q-Horn is incomparable with parameters for CNF-SAT based on width measures (such as the branch-width of the formula's hypergraph [2], treewidth of the formula's primal, dual, or incidence graph [32, 92], or the rank-width of the formula's signed incidence graph [36]). This can be easily verified, since one can define q-Horn formulas where all of these width parameters are arbitrarily large. Conversely, by adding to a formula variable-disjoint copies of itself, we can make the deletion distance from q-Horn arbitrarily large, the width however does not increase.

There are several interesting research questions that arise from the work presented in this chapter. It is an interesting question whether the W[2]-hardness of the detection of strong q-Horn-backdoor sets (Theorem 9.8.1) also holds if the input formula is in 3-CNF. Furthermore, our hardness results contribute additional attention and significance to the problem of whether the parameterized HITTING SET problem has an FPT-approximation algorithm [74].

Part V

Skew-Symmetric Multicuts

10

Skew Symmetric Multicut

10.1 Introduction

A skew-symmetric graph is a digraph $D = (V, A)$ along with an involution $\sigma : V \cup A \to V \cup A$ where for every $x \in V \cup A$, $\sigma(x) \neq x$ and $\sigma(\sigma(x)) = x$ and for every $x \in V$ ($x \in A$), $\sigma(x) \in V$ (respectively $\sigma(x) \in A$). Skew-symmetric graphs were introduced under the name of *antisymmetrical digraphs* by Tutte [96] along with a notion of self-conjugate flows as a generalization of maximum flows in networks and matchings in graphs and subsequently by Zelinka [102] and Zaslavsky [101]. Goldberg and Karzanov [41, 42] revisited the work of Tutte and gave unified proofs for the analogues of the flow-decomposition and max-flow min-cut theorems on these graphs.

In this chapter, we use skew-symmetric graphs and an appropriate notion of separators on them as a model to abstract out "cut properties" underlying several problems in parameterized complexity.

We now introduce the main problem studied in this chapter – a variant of the MULTICUT problem on skew-symmetric graphs.

d-SKEW-SYMMETRIC MULTICUT **Parameter:** k

Input: A skew-symmetric graph $D = ((V, A), \sigma)$, a family $\mathcal{T}$ of d-sets of vertices, integer k.

Question: Is there a set $S \subseteq A$ such that $S = \sigma(S)$, $|S| \leq 2k$, and for any d-set $\{v_1, \ldots, v_d\}$ in $\mathcal{T}$, there is a vertex v_i such v_i and $\sigma(v_i)$ lie in distinct strongly connected components of $D \setminus S$?

The set S is the above definition is called a *skew-symmetric multicut* for the given instance. Our main result is a FPT algorithm for the above problem where the dependence of the running time of the algorithm on the input size is linear. Formally,

Theorem 10.1.1. *There is an algorithm that, given an instance* $(D = (V, A, \sigma), \mathcal{T}, k)$ *of* d-SKEW-SYMMETRIC MULTICUT, *runs in time* $\mathcal{O}((4d)^k k^4(\ell + m + n))$ *and either returns a skew-symmetric multicut of size at most* $2k$ *or correctly concludes that no such set exists, where* $m = |A|$ *and* $n = |V|$, *and* ℓ, *the length of the family* $\mathcal{T}$, *is defined as* $d \cdot |\mathcal{T}|$.

Overview of our algorithm. The main obstacle to applying existing digraph algorithms on skew-symmetric graphs comes from the fact that standard arguments heavily based on submodularity of cuts break down, allowing only approximations, (see for example [78, 38]). Our first contribution is a reduction rule which overcomes this obstacle by allowing us to essentially (and correctly) think of *local* parts of an irreducible instance as a normal digraph. The reduction rule is essentially the following.

"Given two vertex sets X and Y which satisfy certain properties, if there is a minimum X-Y separator which contains an arc a and its image $\sigma(a)$, then some arc (*not necessarily* a) in this separator is part of an optimal solution and therefore, we find this arc, delete it from the instance, reduce the budget and continue."

Though simple to state, the soundness of this reduction rule is far from obvious and requires certain structural observations specific to separators in skew-symmetric graphs. Observe that this is a *parameter decreasing rule* which ensures that the number of applications of this rule is bounded linearly in the parameter. We believe that this rule could prove to be of independent interest for data reduction (or kernelization) for this as well as other similar problems.

Given this reduction rule, the next obstacle we need to overcome is that of applying this rule in linear time. For this, we start from the Ford-Fulkerson algorithm for computing maximum flows and by coupling it carefully with structural properties of skew-symmetric graphs, show that in linear time, we can either apply the reduction rule or locate a part of the graph which is already "reduced" for the next step of our algorithm. In the final step of our algorithm, having found a reduced part of the graph, we show that it is sufficient for us to consider arcs emanating from this part whose number is linearly bounded in the parameter and move ahead by performing an exhaustive branching on this bounded set of arcs. Finally, by a combination of these three subroutines, we obtain a linear time FPT algorithm for d-SKEW-SYMMETRIC MULTICUT.

An additional feature of the algorithm we present is that it does not require the family $\mathcal{T}$ to be explicitly given as part of the input. It is sufficient for our algorithm to have access to a linear time violation oracle for $\mathcal{T}$, i.e, an algorithm which, in linear time returns a violated set in $\mathcal{T}$. This feature increases the utility of this algorithm substantially and we demonstrate this in the case of DELETION q-Horn BACKDOOR SET DETECTION where even though a direct reduction does not run in linear time, we can use a linear time violation oracle to obtain a linear time FPT algorithm for DELETION q-Horn BACKDOOR SET DETECTION.

Applications. Here we provide the list of main applications (see Figure 10.1) that can be derived from our algorithm (Theorem 10.1.1) together with a short overview of previous work on each application.

ODD CYCLE TRANSVERSAL, ALMOST 2-SAT and related problems. Reed et al. [91] showed that ODD CYCLE TRANSVERSAL (ODD CYCLE TRANSVERSAL) is FPT by developing an algorithm for the problem running in time $\mathcal{O}(3^k mn)$ and asked whether there is an FPT algorithm for ODD CYCLE TRANSVERSAL with a linear dependence on the input size. Fiorini et al. [31] showed that when the input is restricted to planar graphs, there is an $\mathcal{O}(f(k)n)$ time algorithm– a linear time FPT algorithm, for ODD CYCLE TRANSVERSAL. Continuing this line of research, recently, Kawarabayashi and Reed [53] obtained an algorithm for ODD CYCLE TRANSVERSAL on general graphs with an improved dependence on the input size. This algorithm uses tools from graph minors and odd variants of graph minors and runs in time $\mathcal{O}(f(k)m \cdot \alpha(m,n))$. Here the function $\alpha(m,n)$ is the inverse of the Ackermann function (see by Tarjan [94]) and $f(k)$ is at least a triple exponential function. However, an algorithm on general graphs with a linear dependence on the input size has so far proved elusive. The work we present in this chapter has led to the first linear time algorithm for ODD CYCLE TRANSVERSAL running in time $\mathcal{O}(4^k k^4(m+n))$.

Recall that in the *edge* version of ODD CYCLE TRANSVERSAL, namely EDGE BIPARTIZATION, the objective is to test if there is a set of at most k edges whose deletion makes the input graph bipartite. Using a known linear time parameter preserving reduction from EDGE BIPARTIZATION to ODD CYCLE TRANSVERSAL, we also get a similar result for EDGE BIPARTIZATION. In fact, both these problems have linear time parameter preserving reductions to the more general problem of ALMOST 2-SAT. In

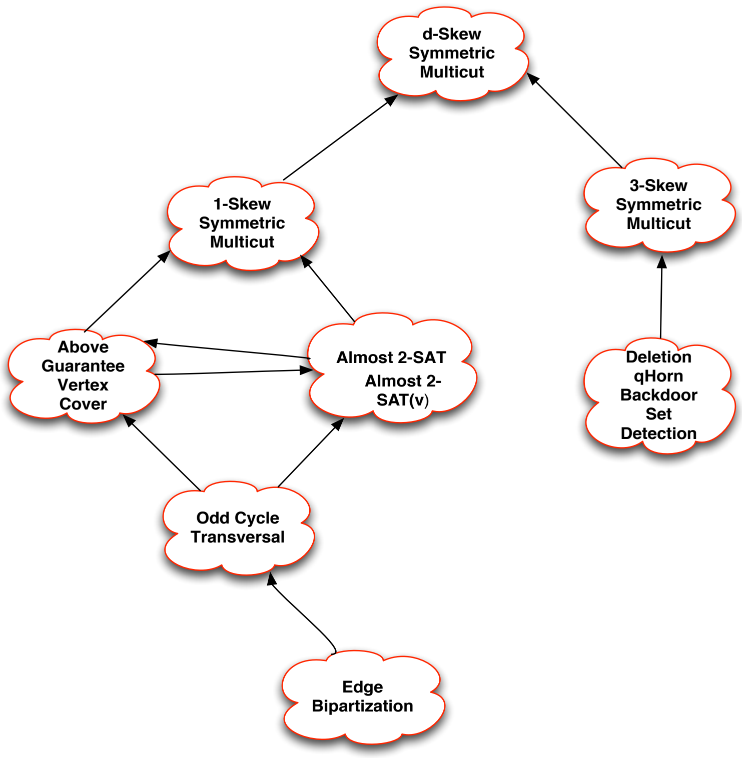
d-Skew Symmetric Multicut
1-Skew Symmetric Multicut
3-Skew Symmetric Multicut
Above Guarantee Vertex Cover
Almost 2-SAT
Almost 2-SAT(v)
Deletion qHorn Backdoor Set Detection
Odd Cycle Transversal
Edge Bipartization

Figure 10.1: Problems with a linear time parameter preserving reduction to d-SKEW-SYMMETRIC MULTICUT

fact, our algorithms for EDGE BIPARTIZATION and ODD CYCLE TRANSVERSAL are obtained via reductions to ALMOST 2-SAT, which in turn is solved using our algorithm for d-SKEW-SYMMETRIC MULTICUT (Theorem 10.1.1).

The ALMOST 2-SAT problem is formally defined as follows.

ALMOST 2-SAT **Parameter:** k

Input: A 2CNF formula F, positive integer k

Question: Does there exist a set S_c of at most k clauses of F such that $F \setminus S_c$ is satisfiable?

The parameterized complexity status of ALMOST 2-SAT remained open until 2008 when Razgon and O'Sullivan [89] gave an algorithm running in time $\mathcal{O}(15^k km^3)$ on formulas with m clauses. More recently, there have been a series of improved algorithms ($\mathcal{O}(9^k n^{\mathcal{O}(1)})$[88], $\mathcal{O}(4^k n^{\mathcal{O}(1)})$[24], $\mathcal{O}(2.618^k n^{\mathcal{O}(1)})$[82]) with the current best algorithm running in time $\mathcal{O}(2.32^k n^{\mathcal{O}(1)})$ [67]. However, none of these algorithms have a linear dependence on the input size. We show that ALMOST 2-SAT is a special case of 1-SKEW-SYMMETRIC MULTICUT, resulting in an algorithm for ALMOST 2-SAT which runs in time $\mathcal{O}(4^k k^4 \ell)$ where k is the size of the solution and ℓ is the length of the input formula.

As discussed earlier, ABOVE GUARANTEE VERTEX COVER (AGVC) is linear time equivalent to ALMOST 2-SAT in a parameter preserving way and hence, our results imply an algorithm for ABOVE GUARANTEE VERTEX COVER with running time $\mathcal{O}(4^{k-|M|}(k-|M|)^4(m+n))$. However, the linear time reduction from ABOVE GUARANTEE VERTEX COVER to ALMOST 2-SAT crucially utilizes the fact that a maximum matching of the graph is also part of the input. Therefore, a natural goal moving forward would to obtain an algorithm running in time $f(k-|M|)(m+n)$ for ABOVE GUARANTEE VERTEX COVER even when a maximum matching is not given as part of the input.

DELETION q-Horn BACKDOOR SET DETECTION and related problems. Recall that a *strong $\mathcal{C}$-backdoor set* of a CNF formula F is a set B of variables such that $F[\tau] \in \mathcal{C}$ for each assignment $\tau : B \to \{0,1\}$, that is, for every instantiation of the variables in B, the reduced formula is in the class $\mathcal{C}$. A *deletion $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F - B \in \mathcal{C}$. Also recall that the DELETION q-Horn BACKDOOR SET DETECTION problem is the following.

DELETION q-Horn BACKDOOR SET DETECTION	**Parameter:** k
Input: A CNF formula F and a positive integer k	
Question: Does F have a deletion q-Horn backdoor set of size at most k?	

We show that DELETION q-Horn BACKDOOR SET DETECTION is a special case of 3-SKEW-SYMMETRIC MULTICUT, giving us an algorithm for DELETION q-Horn BACKDOOR SET DETECTION which runs in time $\mathcal{O}(12^k k^5 \ell)$ where k is the size of the solution and ℓ is the length of the input formula. This improves upon the results of the previous chapter and gives the first fixed-parameter tractable algorithm for this problem. Using this result, we get an algorithm for SATISFIABILITY which runs in time $\mathcal{O}(12^k k^5 \ell)$ where k is the size of the smallest q-Horn deletion backdoor set, with ℓ being the length of the input formula.

Organization of the chapter. In Section 10.2, we define the notions of separators in skew-symmetric graphs, followed by structural results on separators in skew-symmetric graphs and the notions of (L, k)-components whose computation is at the core of our algorithm. In Section 10.3, we prove an observation regarding the structure of optimal solutions, followed by a description and proof of correctness of our algorithm. In Section 10.4, we give linear time parameterized algorithms for a number of problems using our result.

10.2 Skew-symmetric graphs, separators and components

In this section we begin by giving some basic definitions and set up the notations for the rest of the chapter. Following that, we prove some structural results properties regarding separators in skew-symmetric graphs.

Skew-Symmetric Graphs. The notation is from [42]. A *skew-symmetric graph* is a digraph $D = (V, A)$ and an involution $\sigma : V \cup A \to V \cup A$ such that:

1. for each $x \in V \cup A$, $\sigma(x) \neq x$ and $\sigma(\sigma(x)) = s$.

2. for each $v \in V$, $\sigma(v) \in V$

3. for each $a = (v, w) \in A$, $\sigma(a) = (\sigma(w), \sigma(v))$

We call $\sigma(x)$ *symmetric* to x and also refer to x and $\sigma(x)$ as *conjugates*. For ease of description, we let x' denote the conjugate of an element x and we let S' denote the set of conjugates of the elements in the set S. We say that a set S is *regular* if $S \cap S' = \emptyset$ and *irregular* otherwise. A set S is called *self-conjugate* if $S = S'$.

The following observation is a direct consequence of the definition of a skew-symmetric graph.

Observation 10.2.1. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and let $u, v \in V$. There is a path from v to u in D if and only if there is a path from u' to v'.*

Definition 10.2.2. *Let $D = (V, A)$ be a directed graph and let X, Y be disjoint subsets of V. A set $S \subseteq A$ is an X-Y separator if there is no directed path from X to Y in the graph $D \setminus S$. We say that S is a minimal X-Y separator if no proper subset of S is an X-Y separator.*

Definition 10.2.3. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and let L be a regular set of vertices. Let $X \subseteq A$ be a self-conjugate set of arcs of D. We call X an L-L'* **self-conjugate separator** *if X is a (not necessarily minimal) L-L' separator. We call X a minimal L-L' self-conjugate separator if there is no self-conjugate strict subset of X which is also an L-L' separator.*

Definition 10.2.4. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and let L be a regular set of vertices. Let X be an L-L' self-conjugate separator. We denote by $R(L, X)$ the set of vertices of D that can be reached from L via directed paths in $D \setminus X$, and we denote by $\bar{R}(L, X)$ the set of vertices of D which have a directed path to can reach L in $D \setminus X$.*

Observation 10.2.5. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and let L be a regular set of vertices. Let X be an L-L' self-conjugate separator. Then, the sets $R(L, X)$ and $\bar{R}(L', X)$ are also regular and $\sigma(R(L, X)) = \bar{R}(L', X)$.*

Proof. Since deleting a self-conjugate set of arcs from a skew-symmetric graph results in a skew-symmetric graph, we know that there is a path from u to v in $D \setminus X$ if and only if there is a path from v' to u' in $D \setminus X$. Therefore, if $R(L, X)$ is irregular, then there is a path from L to y and y' for some vertex y, which is disjoint from X, which implies a path from L to L' in $D \setminus X$, which is a contradiction. Therefore, $R(L, X)$ and $\bar{R}(L', X)$ are regular and since $D \setminus X$ is a skew-symmetric graph, they are conjugates. □

10.2.1 Minimum separators in skew-symmetric graphs

Lemma 10.2.6. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph, L be a regular set of vertices.*

1. *Suppose that there is an L-L' path in D and let X be a minimum L- L' separator and let $Z = R(L, X \cup X')$. Then, $\delta^+(Z)$ is also a minimum L-L' separator.*

2. *An arc is part of a minimum L-L' separator if and only if its conjugate is also part of a minimum L-L' separator.*

Proof. Recall that Z is regular (Observation 10.2.5). Since L is in Z and L' is disjoint from Z, $\delta^+(Z)$ is an L-L' separator. It remains to show that it is a minimum such separator. Clearly, $\delta^+(Z) \subseteq X \cup X'$. Let $A = \delta^+(Z) \cap X$ and $B = \delta^+(Z) \setminus X$.

Since B is disjoint from X, it must be the case that $B' \subseteq X$. We now claim that A and B' are disjoint. Suppose that this is not the case and let $x \in B$ such that $x' \in A$. Since $x' \in \delta^+(Z)$, it must be the case that $x \in \delta^-(Z')$. Since there is a path from Z to Z' via x and X is disjoint from $A[Z \cup Z'] \cup \{x\}$, there is a path from L to L' disjoint from X, a contradiction. Therefore, we conclude that $A \cap B' = \emptyset$. We now have that $|\delta^+(Z)| = |A \cup B| = |A| + |B'| \leq |X|$ where the last inequality used the fact that A and B' are disjoint. Therefore, we conclude that $\delta^+(Z)$ is indeed a minimum L-L' separator. Consequently, $\delta^-(Z)$ is also a minimum L-L' separator. This concludes the proof of the first part of the lemma.

We claim that if X is an L-L' separator, then X' is an L-L' separator as well. Suppose that this is not the case and let there be a path $v_1, \ldots, v_r$ in $D \setminus X'$ where $v_1 = l$ and $v_r = l'$. However, this implies that X is disjoint from the path $\sigma(v_r), \ldots, \sigma(v_1)$, which is a contradiction. This concludes the proof of the lemma. $\square$

Lemma 10.2.7. *(Crossing-Uncrossing) Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and let $B \subseteq V$. If $\delta^+(B) \cap \delta^+(B') = \emptyset$ then $|\delta^+(B \setminus B')| = |\delta^+(B)|$ and $|\delta^+(B \setminus B')| < |\delta^+(B)|$ otherwise.*

Proof. Let $Q = B \setminus B'$. We partition $\delta^+(Q)$ into the following sets (see Figure 10.2).

1. $Q_1^o = \delta^+(Q) \cap \delta^-(V \setminus (B \cup B'))$, that is, those arcs with the tail in Q and the head in $V \setminus (B \cup B')$.
2. $Q_2^o = \delta^+(Q) \cap \delta^-(B \setminus B')$, that is, those arcs with the tail in Q and the head in $B' \setminus B$.
3. $Q_3^o = \delta^+(Q) \cap \delta^-(B \cap B'))$, that is, those arcs with the tail in Q and the head in $B \cap B'$.

Similarly, we partition $\delta^+(B)$ as follows.

1. $B_1^o = (\delta^+(B \setminus B') \cap \delta^-(V \setminus (B \cup B'))$, that is, those arcs with the tail in $B \setminus B'$ and the head in $V \setminus (B \cup B')$.
2. $B_2^o = (\delta^+(B) \cap \delta^-(B' \setminus B)$, that is, those arcs with the tail in $B \setminus B'$ and the head in $B' \setminus B$.

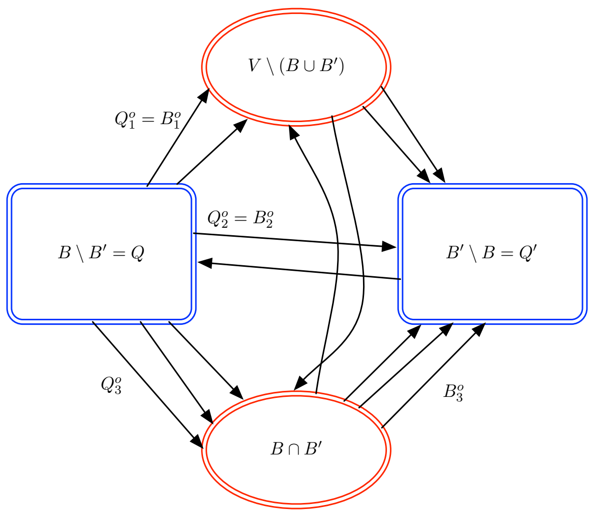

Figure 10.2: An illustration of the partitions described in the proof of Lemma 10.2.7

3. $B_3^o = \delta^+(B) \cap \delta^+(B' \setminus B)$, that is, those arcs with the tail in $B \cap B'$ and the head in $B \setminus B'$.
4. $B_4^o = \delta^+(B) \cap \delta^+(B')$

Observe that $Q_1^o = B_1^o$, $Q_2^o = B_2^o$ and $Q_3^o = (B_3^o)'$. Therefore, $|\delta^+(Q)| = |\delta^+(B)|$ if B_4^o is empty and $|\delta^+(Q)| < |\delta^+(B)|$ otherwise. This completes the proof of the lemma. $\square$

10.2.2 (L, k)-Components

Definition 10.2.8. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and $k \in \mathbb{N}$. Let $L \subseteq V$ be a regular set of vertices. A set of vertices $Z \subseteq V$ is called an (L, k)-component if it satisfies the following properties.*

1. *$L \subseteq Z$*
2. *Z is regular*
3. *Z is reachable from L in $D[Z]$*

4. *The size of a minimum Z-Z' separator is equal to the size of a minimum L-L' separator and this size is at most $2k$.*

5. *Z is inclusion-wise maximal among the sets satisfying the above properties.*

Lemma 10.2.10 gives an algorithm that in linear time either computes an (L, k)-component or finds a minimum L-L' separator of a particular kind, which we later show can be used to reduce the instance. For this, we require the following lemma which we proved in Chapter 3.

Lemma 10.2.9. *Let s, t be two vertices in a digraph $D = (V, A)$ such that the minimum size of an s-t separator is $\ell > 0$. Then, there is a collection $\mathcal{X} = \{X_1, \dots, X_q\}$ of sets where $\{s\} \subseteq X_i \subseteq V \setminus \{t\}$ such that*

1. *$X_1 \subset X_2 \subset \cdots \subset X_q$,*
2. *X_i is reachable from s in $D[X_i]$,*
3. *$|\delta^+(X_i)| = \ell$ for every $1 \leq i \leq q$ and*
4. *every s-t separator of size ℓ is fully contained in $\bigcup_{i=1}^{q} \delta^+(X_i)$.*

Furthermore, there is an $\mathcal{O}(\ell(|V|+|A|))$ time algorithm that produces the sets $X_1, X_2 \setminus X_1, \dots, X_q \setminus X_{q-1}$ corresponding to such a collection $\mathcal{X}$.

Using the above lemma, we prove the main lemma of this section, which is the following.

Lemma 10.2.10. *Let $D = ((V, A), \sigma)$ be a skew-symmetric graph and $k \in \mathbb{N}$. Let $L \subseteq V$ be a regular set of vertices such that there is an L-L' path in D. There is an algorithm which runs in time $\mathcal{O}(k^3(m + n))$ and*

- *correctly concludes that no (L, k)-component exists or*

- *returns an (L, k)-component or*
- *returns an irregular minimum L-L' separator*

where $m = |A|$ and $n = |V|$.

Proof. The main idea of the algorithm is to start with the collection coming from Lemma 3.2.27 and then use this to either find an (L, k)-component or to return an irregular minimum L-L' separator. In what follows we describe possible situations that could arise and how they could be handled. Finally, we use all this to describe the algorithm and prove its correctness.

We can, in $\mathcal{O}(k(m + n))$ time check if the size of the minimum L-L' separator is at most $2k$ by running $2k$ iterations of the Ford-Fulkerson algorithm [35]. If the size of the minimum separator exceeds $2k$, then we can conclude that no (L, k)-component exists. Therefore, we assume in the rest of this proof that the size of the minimum L-L' separator is at most $2k$. Let $\mathcal{X} = \{X_1, \ldots, X_q\}$ be a collection with the properties mentioned in Lemma 3.2.27 where "L acts as s and L' acts as t". We make this formal when we describe the algorithm later.

We begin by showing that not all X_i's can be irregular.

Claim 8. *There is an index $i \geq 1$ such that for all $j \leq i$, the set X_j is regular.*

Proof. We first show that X_1 is regular. Suppose that this is not the case and there are vertices $y, y' \in X_1$. Since no arc in $A[X_1]$ is part of a minimum L-L' separator (by property 4 of the collection), Lemma 10.2.6 implies that no arc in $A[X_1]$ is the conjugate of an arc in $S = \delta^+(X_1)$. Therefore, there is a path from L to y and a path from L to y' disjoint from $S \cup S'$. However, this implies the presence of a path from L to L' in $D \setminus (S \cup S')$, which is a contradiction. Therefore, X_1 is regular. Since every subset of a regular set is regular, there is an index $i \geq 1$ such that for all $j \leq i$, X_j is regular. This completes the proof of the claim. $\square$

Given the above claim, we first consider the case when X_i is regular for every $1 \leq i \leq q$. This brings us to the following claim.

Claim 9. *If X_i is regular for every $1 \leq i \leq q$ then X_q itself is an (L, k)-component.*

Proof. Observe that X_q satisfies the first four properties of an (L, k)-component and thus it suffices to prove that X_q is inclusion-wise maximal with respect to these 4 properties. Suppose that X_q is not maximal with respect to these properties and let $Z \supset X_q$ have the required properties and let Y be a minimum Z-Z' separator. Clearly, Y is a minimum L-L' separator. Since $Z \supset X_q$, there is an arc $y \in Y$ which is not in $\delta^+(X_q)$. Since X_q strictly contains all other X_i's, $y \notin \delta^+(X_i)$ for any i, which contradicts property 4 of the collection. $\square$

Claims 8 and 9 act like base cases of our algorithm that we describe later.

We now suppose that there exists $i \leq q$ such that X_i is irregular. Let a be the highest index such that X_a is regular and X_{a+1} is irregular. Let $A = X_a$ and $B = X_{a+1}$. Since $\delta^+(B)$ is a minimum L-L' separator, by the crossing-uncrossing lemma (Lemma 10.2.7), we have that $|\delta^+(B \setminus B')| = |\delta^+(B)|$ and $\delta^+(B) \cap \delta^+(B') = \emptyset$. That is, there is no arc which enters $V \setminus (B \cup B')$ from $B \cap B'$ and thus there is no arc which enters $B \cap B'$ from $V \setminus (B \cup B')$. Furthermore, if there is an arc $x \in \delta^+(B \setminus B') \cap \delta^-(B' \setminus B)$, then $x' \in \delta^+(B \setminus B')$, which implies that x, x' are contained in a minimum L-L' separator. Therefore, from this point on, we may assume that there is no arc in $\delta^+(B \setminus B') \cap \delta^-(B' \setminus B)$. Before we go further we summarize the sets and the various intersections they have. From now onwards the sets B and $Q = B \setminus B'$ will always have the following intersection properties.

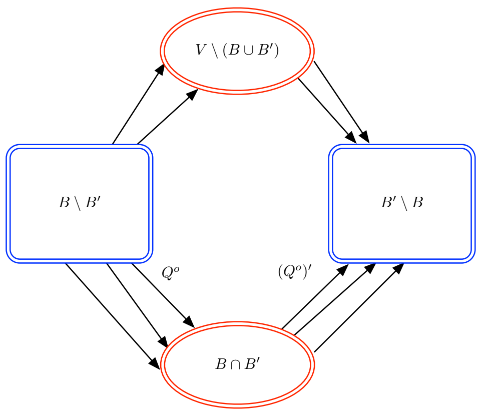

Figure 10.3: An illustration of the sets defined in Lemma 10.2.10. Observe that there are no arcs between $B \setminus B'$ and $B' \setminus B$ and between $B \cap B'$ and $V \setminus (B \cup B')$

1. $Q = B \setminus B'$
2. $|\delta^+(B \setminus B')| = |\delta^+(Q)| = |\delta^+(B)|$
3. $\delta^+(B) \cap \delta^+(B') = \emptyset$
4. $\delta^-(B) \cap \delta^-(B') = \emptyset$
5. $\delta^+(B \setminus B') \cap \delta^-(B' \setminus B) = \delta^+(Q) \cap \delta^-(Q') = \emptyset$

The proof of the next observation follows from the fact that there is neither an arc entering $B \cap B'$ from $V \setminus (B \cup B')$ nor an are leaving $B \cap B'$ to $V \setminus (B \cup B')$ (see Figure 10.3).

Observation 10.2.11. *Any path from Q to Q' with the internal vertices disjoint from $Q \cup Q'$ is contained entirely in either $D \setminus (B \cap B')$ or $D[B \cup B']$.*

The next claim describes certain properties of paths exiting Q and entering $B \cap B'$. This will be used later in some of the arguments.

Claim 10. *Let B and Q be defined as above. Then for every $q \in N^+(Q) \cap (B \cap B')$ there is a path from q to $N^-(Q') \cap (B \cap B')$ which completely lies in the graph $D[B \cap B']$.*

Proof. Suppose this is not the case. That is, there exists an edge $x = (p, q) \in \delta^+(Q)$ such that $q \in (B \cap B')$ and there is no path from q to $N^-(Q') \cap (B \cap B')$ which completely lies in the graph $D[B \cap B']$. Consider the graph $D_1 = D \setminus (\delta^+(Q) \setminus \{x\})$. Since $\delta^+(Q)$ is a minimum L-L' separator, there is path from L to L' in D_1. Since this path contains the arc x, there is a subpath, say W, from q to $N^-(Q')$ that does not intersect Q'. Furthermore, in D_1 the only arc that emanates from Q is x and thus we have that W is disjoint from Q as well. However, by Observation 10.2.11 every path from Q to Q' with the internal vertices disjoint from Q and Q' is contained in $D[B \cup B']$ or $D[V \setminus (B \cap B')]$. Since $q \in B \cap B'$, we conclude that there is a path from q to $N^-(Q') \cap (B \cap B')$ in $D[B \cap B']$. This is a contradiction to our assumption and thus concludes the proof. $\square$

We are now ready to describe the cases that occur when we *have* an irregular set. This case is divided into following three exhaustive subcases.

Case I: $A \cap B' = \emptyset$.

Case II: $A \cap B' \neq \emptyset$ and $A \setminus B' \neq \emptyset$.

Case III: $A \subseteq B \cap B'$.

We now consider each case one by one and show how we will handle it algorithmically (later).

Case I: $A \cap B' = \emptyset$. We start by defining the set $Q^o = \delta^+(Q) \cap \delta^-(B \cap B')$.

The next claim proves an interesting structural property that is crucial to the correctness of the algorithm.

Claim 11. *Either every (L,k)-component $Z \supseteq Q$ is such that $Q^o \subseteq \delta^+(Z)$ or there is an arc $y \in Q^o$ such that there is a minimum L-L' separator containing y and y'.*

Proof. Let $Z \supseteq Q$ be an (L,k)-component and $x \in Q^o$ an arc such that $x = (p,q) \notin \delta^+(Z)$. Note that this implies that $x \in A[Z]$. By Claim 10, we have that there is a path from q to $N^-(Q') \cap (B \cap B')$ which lies in the graph $D[B \cap B']$. Observe that $N^-(Q') \cap (B \cap B') = \textbf{Tail}((Q^o)')$. Let $r \in B \cap B'$ be such that q has a path in $D[B \cap B']$ from q to r, where $(r,s) \in (Q^o)'$. We claim that there is a minimum L-L' separator containing (r,s) and (s',r'). Consider a minimum Z-Z' separator S. Since every arc in $A[B \cap B']$ has both endpoints inside B and both endpoints outside A, none of these arcs appear in the set $\bigcup_{i=1}^{q} \delta^+(X_i)$, we conclude that S is disjoint from $A[B \cap B']$ (by property 4 of the collection). Since S is disjoint from $A[B \cap B']$, we have that $r \in Z$. Furthermore, since $s \in Q$, we have that $s \in Z'$. Consequently, we have that $r' \in Z'$ and $s' \in Z$. Therefore, both the arcs (r,s) and (s',r') are in both the sets $\delta^+(Z)$ and $\delta^-(Z')$, which implies that they are also present in S. Since S is also a minimum L-L' separator, this completes the proof of the claim. $\square$

The following claims allow our algorithm to use the above observations recursively in the graph $D \setminus (B \cap B')$ in the absence of $y \in Q^o$ such that there is no minimum L-L' separator containing y and its conjugate y'.

Claim 12. *Assume that every (L,k)-component $Z \supseteq Q$ is such that $Q^o \subseteq \delta^+(Z)$. (Recall $Q^o = \delta^+(Q) \cap \delta^-(B \cap B')$.)*

1. *If S is a minimum Q-Q' separator in $D_1 = D[V \setminus (B \cap B')]$, then $S \cup Q^o$ is a minimum Q-Q' separator in D.*
2. *If Z is an (L,k)-component such that $Q^o \subseteq \delta^+(Z)$, then Z is also an $(L, k-|Q^o|)$-component in $D_1 = D[V \setminus (B \cap B')]$ and furthermore, any $(L, k-|Q^o|)$-component*

in D_1 is an (L,k)-component in D.

Proof. Recall that $\delta^+(B) \cap \delta^+(B') = \emptyset$ and $\delta^-(B) \cap \delta^-(B') = \emptyset$. Hence every Q-Q' path in D is contained entirely in the graph D_1 or $D_2 = D[B \cup B']$. The last assertion implies that the size of the minimum Q-Q' separator in D is the sum of the sizes of the minimum Q-Q' separator in the graphs D_1 and D_2. Since $\delta^+(Q) \setminus Q^o$ is a minimum Q-Q' separator in D_1, S is no larger than $\delta^+(Q) \setminus Q^o$. Therefore, $S \cup Q_o$ is no larger than $\delta^+(Q)$. Since $S \cup Q^o$ is clearly a Q-Q' separator in D, this completes the proof of this statement.

Now we show the second part of the claim. We first show that Z satisfies the first 4 properties of an $(L, k - |Q^o|)$-component in D_1. Recall that the size of a minimum Q-Q' separator in the graph D_1 is $|\delta^+(Q)| - |Q^o|$, which implies that Z indeed satisfies the first 4 properties of an $(L, k - |Q^o|)$-component in D_1.

Therefore, if Z were not an $(L, k - |Q^o|)$-component in D_1, then there is a set $W \supset Z$ which satisfies these 4 properties and let S be a minimum W-W' separator in D_1. We claim that W also satisfies the first 4 properties of an (L,k)-component in D, which contradicts our assumption of Z as an (L,k)-component. From the first part of the claim, we have that $S \cup Q^o$ is a minimum Q-Q' separator in D, which implies that W indeed satisfies the first 4 properties of an (L,k)-component in D.

In the converse direction, let Z be an $(L, k - |Q^o|)$-component in D_1. Since $S \cup Q^o$ is a minimum Q-Q' separator in D, we have that Z satisfies the first 4 properties of an (L,k)-component in D. For contradiction assume that Z is not an (L,k)-component in D. Then there exists $W \supset Z$ that is a (L,k) component in D. Since every (L,k)-component W is such that $Q^o \subseteq \delta^+(W)$, we have that $W \cap \textbf{Head}(Q^o) = \emptyset$. But then it implies that W also satisfies the first 4 properties of an $(L, k - |Q^o|)$-component in D_1. However this is a contradiction to our assumption that Z is an $(L, k - |Q^o|)$-component in D_1. This completes the proof of the claim. $\square$

This completes the study of the case when $A \cap B' = \emptyset$.

Case II : $A \cap B' \neq \emptyset$ and $A \setminus B' \neq \emptyset$.

Here, for a subcase we construct a new collection $\mathcal{X}'$ where this case is avoided. Let $Q = B \setminus B'$ and $P = A \cup (B \setminus B')$. We have already observed that $|\delta^+(Q)| = |\delta^+(B)|$ and that the set $\delta^+(B) \cap \delta^+(B')$ is empty. We now show that $|\delta^+(Q)| = |\delta^+(P)|$. Let $P_1^o = \delta^+(P) \setminus \delta^+(Q)$ and $Q_1^o = \delta^+(Q) \setminus \delta^+(P)$. We claim that $|P_1^o| = |Q_1^o|$. But this is true since $|P_1^o| < |Q_1^o|$ implies that $|\delta^+(P)| < |\delta^+(Q)|$, which is a contradiction since $\delta^+(P)$ is an L-L' separator smaller than the minimum one and $|Q_1^o| < |P_1^o|$ implies that $|\delta^+(A \setminus B')| < |\delta^+(A)|$, which is a contradiction since $\delta^+(A \setminus B)$ is an L-L' separator smaller than the minimum one. Therefore, we conclude that $|P_1^o| = |Q_1^o|$ and hence $|\delta^+(Q)| = |\delta^+(P)|$.

By combining this along with the crossing-uncrossing lemma (Lemma 10.2.7) applied on the set $K = B \setminus A'$, we have that $|\delta^+(P)| = |\delta^+(K \setminus K')| = |\delta^+(K)|$. Furthermore, $A \subset K \subset B$, and $A \cap K' = \emptyset$.

If K is irregular, then consider the collection $\mathcal{X}'$ obtained by inserting the set K between X_a and X_{a+1} in the collection $\mathcal{X}$. Clearly, $\mathcal{X}'$ is also a collection which satisfies the properties 1, 2, and 4 of Lemma 3.2.27. It also satisfies property 3 since $\delta^+(K)$ is a minimum L-L' separator. However, if we consider the collection $\mathcal{X}'$, A would still be the regular set with the highest index, while K would be the irregular set with the least index. Since A is disjoint from K', we fall back into the previous case when we consider this collection.

If K is regular, then $(B \cap B') \setminus (A \cup A') = \emptyset$. Observe that $N^+(Q) \cap (B \cap B') \subseteq A \cap B'$ and $N^-(Q') \cap (B \cap B') \subseteq A' \cap B$. Since $A \cap B' \neq \emptyset$ and every vertex in A is reachable from L, we have that $N^+(Q) \cap (B \cap B') \neq \emptyset$ and thus $N^-(Q') \cap (B \cap B') \neq \emptyset$. This together with Claim 10 implies that there is a path, say W, from $A \cap B'$ to $A' \cap B$ which lies in the graph $D[B \cap B']$. Let p be the last vertex on W from $A \cap B'$. Since $A \cap B'$ and $A' \cap B$ partition $B \cap B'$, either there is an arc (p, q) such that $p \in A \cap B'$ and $q \in A' \cap B$ or $p \in A \cap B'$ and $q \in B'$. In either case this implies that there is an arc (q', p') where $q' \in A \cap B'$ in the former case and $q \in B'$ in the latter case and $p' \in A' \cap B$. Since both (p, q) and (q', p') are in $\delta^+(K) \cap \delta^-(K')$, and $\delta^+(K)$ is a minimum L-L' separator, we have that $\delta^+(K)$ is a minimum L-L' separator containing arcs y and y' where $y = (p, q)$.

Case III: $A \subseteq B \cap B'$. This case is non-existent since $L \subseteq A$ and $L \cap B' = \emptyset$.

The Algorithm We begin by applying the Ford-Fulkerson algorithm to compute a minimum L-L' separator in the graph. If we require more than $2k$ iterations of the Ford-Fulkerson algorithm, then we return that there is no (L, k)-component. We then apply the algorithm of Lemma 3.2.27 to compute the collection $\mathcal{X} = \{X_1, \ldots, X_q\}$ which can be computed in time $\mathcal{O}(k(m + n))$. In order to apply Lemma 3.2.27, we need vertices s and t. Therefore, we add vertices s and t to the graph, add $2k + 1$ arcs from s to each vertex in L and $2k + 1$ arcs from each vertex in L to t. It is clear from this construction that no s-t separator of size at most $2k$ will contain any of these newly added arcs and therefore, the s-t separators of size at most $2k$ are in one to one correspondence with the L-L' separators of size at most $2k$. This allows us to use Lemma 3.2.27 in the form it is stated in.

We then simply need to examine each $X_{i+1} \setminus X_i$ once to compute the index a such that X_a is the highest regular set and X_{a+1} is the least irregular set. After computing the index a, in $\mathcal{O}(m)$ time, we can compute the case we are currently in by computing the intersection of the sets A and B. If we are in case (b), in time $\mathcal{O}(k^2 m)$, we can compute K and either find an irregular minimum L-L' separator by testing if there is a minimum L-L' separator containing y, y' for some $y \in \delta^+(K)$ or move to case (a) where we already have computed the required sets– the regular set with the highest index A and the irregular set with the least index K. Finally, if we are in case (a), in $\mathcal{O}(m)$ time, we compute the set Q and iteratively compute a (Q, k')-component in $D \setminus (B \cap B')$ (which we have already shown is an (L, k)-component) where $k' < k$ or an irregular minimum Q-Q' separator. In the latter case, we add Q^o to this minimum separator to get an irregular minimum Q-Q' separator in D. The correctness of our algorithm follows from the structural claims preceding the description. Furthermore, each time we iterate, we attempt to compute an (L, k')-component where $k' < k$. Therefore, we can have at most $2k$ such iterations and hence, the running time of our algorithm is bounded by $\mathcal{O}(k^3(m + n))$. This completes the proof of the lemma. □

10.3 Algorithm for d-Skew-Symmetric Multicut

In this section we design our linear time parameterized algorithm for d-Skew-Symmetric Multicut. We first give a lemma which allows us to find a solution that is disjoint from some part of the solution. More formally we show the following.

Lemma 10.3.1. *Let $(D = (V, A), \sigma, \mathcal{T}, k)$ be a* Yes *instance of d-*Skew-Symmetric Multicut *and let L be a regular set of vertices such that there is an L-L' path in D. If there is a solution for the given instance which is an L-L' self-conjugate separator in D, then the following hold.*

1. *An (L, k)-component exists.*
2. *Let $Z \subseteq V$ be a regular set of vertices containing L such that the size of the minimum Z-Z' separator is the same as the size of the minimum L-L' separator.*

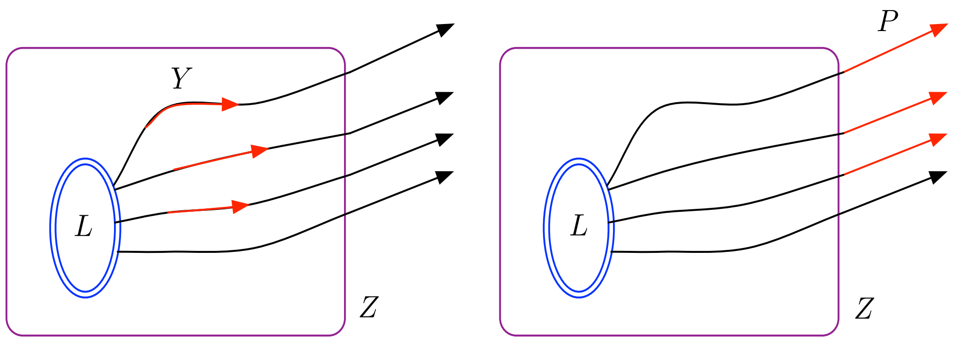

Figure 10.4: An illustration of the sets Z, Y and P in the proof of Lemma 10.3.1.

Then, there is a solution for the given instance disjoint from $A[Z]$.

Proof. For the first statement, observe that since there is a solution for the given instance which is an L-L' self-conjugate separator, the size of the minimum L-L' separator is at most $2k$. Therefore the set L itself satisfies the first 4 properties of an (L, k)-component and therefore an (L, k)-component exists. This completes the proof for the first statement.

Now we prove the second part of the lemma. Let X be the solution defined above, that is, X is an L-L' self-conjugate separator. If X is disjoint from $A[Z]$, then we are done. Therefore, suppose that X intersects $A[Z]$ and let Y be the set of arcs in $A[Z]$ such that $Y \cup Y' \subseteq X$. Let $P \subseteq \delta^+(Z)$ be such that **Tail**(P) is not reachable from L in $D[Z] \setminus Y$. That is, Y is an L-**Tail**(P) separator in the digraph $D[Z]$. It might be possible that **Tail**$(P) = \emptyset$. We now claim that the set $\widehat{X} = (X \setminus (Y \cup Y')) \cup (P \cup P'))$ (see Figure 10.4) is also a solution for the given instance.

Claim 13. $\widehat{X} = (X \setminus (Y \cup Y')) \cup (P \cup P'))$ *is a solution for the given instance and* $|\widehat{X}| \leq |X|$.

Proof. We first show that $|\widehat{X}| \leq |X|$. Since $\delta^+(Z)$ is a minimum L-L' separator, there are $|\delta^+(Z)|$ arc disjoint paths from L to **Tail**$(\delta^+(Z))$ in $D[Z]$. Therefore, $|Y| \geq |P|$. Clearly, $\widehat{X}$ is no larger than X. Therefore, it remains to show that $\widehat{X}$ is a skew-symmetric multicut for D. If this were not the case, then there is a closed walk in the graph $D \setminus \widehat{X}$ intersecting an arc $y \in Y$ and containing vertices t and t'. Here $t \in J$ where $J \in \mathcal{T}$ and for every vertex $v_i \in J$, v_i and $\sigma(v_i)$ lie in the same strongly connected components of $D \setminus \widehat{X}$. That is, J is a violated constraint.

Since **Tail**(y) is reachable from L in the graph $D \setminus \widehat{X}$, both t and t' are reachable from L in the graph $D \setminus \widehat{X}$, which implies the presence of a path, say W, from L to L' in $D \setminus \widehat{X}$. Now using this path we construct another path W' from L to L' in $D \setminus X$. This will contradict our assumption that X is an L-L' self-conjugate separator in D.

Observe that W must also intersect an arc in $\delta^+(Z)$ since $L \subseteq Z$ and L' is disjoint from Z. However, since $P \cup P' \subseteq \widehat{X}$, this arc, say (p, z_1), is in $\delta^+(Z) \setminus P$. Furthermore, this path also contains a subpath from a vertex $z_1 \in$ **Head**$(\delta^+(Z) \setminus P)$ to a vertex $z_2 \in$ **Tail**$(\delta^-(Z') \setminus P')$ whose arcs are disjoint from $A[Z \cup Z'] \cup \delta^+(Z) \cup \delta^-(Z')$. We call this path $W_{z_1 z_2}$. Let (z_2, q) be an arc in $\delta^-(Z') \setminus P'$ such that the arc (z_2, q) is on W. Observe that p, q' are in Z and there are paths from L to both p and q' that avoids arcs of Y. The last assertion follows from the fact that the vertices of $\delta^+(Z) \setminus P$ are reachable from L in $D[Z] \setminus Y$. Let these paths avoiding arcs of Y be called W_{Lp} and $W_{Lq'}$. Then observe that $W_{Lp}W_{z_1 z_2}(W_{Lq'})'$ forms a path in $D \setminus X$ – a contradiction. Here, $(W_{Lq'})'$ is the path that is conjugate to $W_{Lq'}$. This completes the proof of the claim. $\square$

The above claim completes the proof of the lemma. $\square$

From this point on, we assume that an instance of d-SKEW-SYMMETRIC MULTICUT is of the form $(D = (V, A), \sigma, T, k, L)$ where L is a regular set of vertices and the question is to check *if there is a solution for the given instance which is an L-L' self-conjugate separator.* To solve the problem on the given input instance, we simply solve it on the instance $(D = (V, A), \sigma, \mathcal{T}, k, \emptyset)$.

Lemma 10.3.2. *Let $(D = (V, A), \sigma, \mathcal{T}, k, L)$ be an instance of d-Skew-Symmetric Multicut and let S be an irregular minimum L-L' separator. Then, there are arcs y, y' such that $y, y' \in \delta^+(R(L, S \cup S'))$ and there is a solution for the given instance containing y, y'.*

Proof. Let $Z = R(L, S \cup S')$. By Lemma 10.2.6, $\delta^+(Z)$ is a minimum L-L' separator and hence $|\delta^+(Z)| = |S|$. Since S is irregular, $S \cup S'$ contains arcs from at most $|S| - 1$ conjugate pairs. Thus there are arcs $y, y' \in \delta^+(Z)$.

Let X be a solution for the given instance. Since there is no path from L-L' in $D \setminus X$, it must be the case that $D \setminus X$ cannot contain paths from L to both **Tail**(y) and **Tail**(y'). Therefore, it must be the case that X intersects $A[Z]$ and intersects all paths from L to **Tail**(y) or **Tail**(y'). However, by Lemma 10.3.1 we know that there exists a solution that does not intersect $A[Z]$. This implies that there is also a solution containing the arcs y, y'. □

The above lemma gives us the following reduction rule.

Reduction Rule 1. *Let $(D = (V, A), \sigma, T, k, L)$ be an instance of d-Skew-Symmetric Multicut and let S be an irregular minimum L-L' separator. Let y, y' be the arcs given by Lemma 10.3.2. Then return the instance $(D = (V, A \setminus \{y, y'\}), \sigma, \mathcal{T}, k - 1)$.*

Therefore, by combining this reduction rule with the algorithm of Lemma 10.2.10, we can, in linear time either reduce the parameter or compute an (L, k)-component with a regular neighborhood. We are now ready to prove Theorem 10.1.1 by giving an algorithm for d-Skew-Symmetric Multicut.

Description of Algorithm. The input to our algorithm for d-Skew-Symmetric Multicut is an instance $(D = (V, A), \sigma, \mathcal{T} = \{J_1, \ldots, J_r\}, k, L)$ where $J_i = \{v_{i_1}, \ldots, v_{i_d}\}$

and the algorithm either returns a skew-symmetric multicut of size at most $2k$ which is an L-L' self-conjugate separator in D or concludes correctly that no such set exists. In order to solve the problem on the given instance of d-SKEW-SYMMETRIC MULTICUT, the algorithm is invoked on the input $(D = (V, A), \sigma, \mathcal{T}, k, \emptyset)$.

1. If $L = \emptyset$ or if there is no path from L to L' in D, then the algorithm checks if there is a set $J_i \in \mathcal{T}$ such that for all $1 \leq s \leq d$, v_{i_s} and v'_{i_s} lie in the same strongly connected component in D, that is, a violated set. If there is no such set, then the algorithm returns the emptyset. Otherwise, the algorithm picks such a set J_i, and branches in $2d$ ways. In the first d branches, it recurses on the instances $\{(D = (V, A), \sigma, \mathcal{T}, k, \{v_{i_j}\})\}_{1 \leq j \leq d}$ and in the next d branches, it recurses on the instances $\{(D = (V, A), \sigma, \mathcal{T}, k, \{v'_{i_j}\})\}_{1 \leq j \leq d}$.

2. Suppose $L \neq \emptyset$ and there is an L-L' path in D. Then, the algorithm of Lemma 10.2.10 is first used on the instance $(D = (V, A), \sigma, \mathcal{T}, k, L)$ to either compute an (L, k)-component (if it exists) or an irregular minimum L-L' separator. If an (L, k)-component does not exist, then we return NO. If an irregular minimum L-L' separator is returned, then we apply Reduction Rule 1. Suppose that an (L, k)-component Z is returned. We check if Reduction Rule 1 applies on any arc in $\delta^+(Z)$ and if it does, apply the rule. Therefore, at this point, we may assume that an (L, k)-component Z is returned and that the rule is not applicable on any arc in $\delta^+(Z)$. Observe that $\delta^+(Z) \neq \emptyset$ since there is an L-L' path in D. The algorithm then picks an arc $a \in \delta^+(Z)$ and branches in 2 ways as follows. In the first branch, the algorithm deletes $\{a, a'\}$ and recurses on the resulting instances, that is, the algorithm recurses on the instance $(D = (V, A \setminus \{a, a'\}), \sigma, \mathcal{T}, k - 1, L)$. In the next branch, the algorithm recurses on the instance assuming that a is in $A[R(L, X)]$ where X is the hypothetical solution, that is, the algorithm recurses on the instance $(D = (V, A), \sigma, \mathcal{T}, k, L \cup \{\textbf{Head}(a)\})$.

Correctness. The correctness of the algorithm is proved by induction on a measure defined on the instance I. Let this measure be denoted by $\mu(I) = \mu((D,k),L) = 2k - \lambda(L,L')$ where $\lambda(L,L')$ is the size of the smallest L-L' separator. In the base case, if $\lambda(L,L') > 2k$, then the algorithm of Lemma 10.2.10 returns NO on input $(D = (V,A),\sigma,k,L)$, and hence this algorithm returns NO as well, which is correct since the solution we require contains an L-L' separator of size at most $2k$. Similarly, the case when $k < 0$ is also clearly correct. We now assume as induction hypothesis that the algorithm is correct on all instances I such that $\mu(I) \leq \mu - 1$ and consider an instance $I = (D = (V,A),\sigma,\mathcal{T},k,L)$ such that $\mu(I) = \mu$ and $k \geq 0$.

We first show that an application of Reduction Rule 1 does not decrease this measure. Since deleting an arc and its conjugate from an irregular minimum L-L' separator reduces the size of the minimum size L-L' separator by 2 and the budget k by 1, the measure $2k - \lambda(L,L')$ remains unchanged.

We now consider the branching rules. If $L = \emptyset$ or there is no L-L' path in D, then $\lambda(L,L') = 0$. Consider an instance $I' = (D = (V,A),\sigma,\mathcal{T},k,\{v\})$ resulting from a branch here. Although the parameter has not decreased here, since there is a path from v to v', $\lambda(v,v') > 0$, which implies that $\mu(I') < \mu(I)$. Therefore, by combining the exhaustiveness of the branching along with the induction hypothesis, we obtain the correctness of the algorithm on the instance I as well.

We now suppose that $L \neq \emptyset$ and there is an L-L' path in D. The branching is exhaustive due to Lemma 10.3.1. We now show that for each of the resulting instances I' from any of the branches, $\mu(I') \leq \mu - 1$ in which case we can apply the induction hypothesis on these instances, thus proving the correctness of the algorithm.

(a) We begin with the branch where we recurse on the instance $I' = (D = (V,A),\sigma,\mathcal{T},k,Z \cup \{\textbf{Head}(a)\})$. Since Z is an (L,k)-component, by definition, the size of the smallest sep-

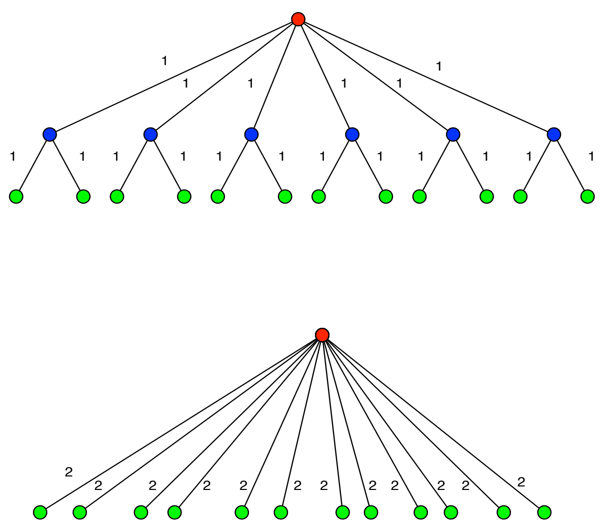

Figure 10.5: An illustration of the tighter analysis of the search tree in the algorithm.

arator from $Z \cup \{\textbf{Head}(a)\}$ to $Z' \cup \{(\textbf{Head}(a))'\}$ is strictly larger than $\lambda(L, L')$, which implies that $\mu(I') < \mu(I)$.

(b) We now consider the instance resulting from the other branch, that is the instance $I' = (D = (V, A \setminus \{a, a'\}), \sigma, \mathcal{T}, k-1, L)$ where $a \in \delta^+(Z)$. In this case, the parameter has decreased by 1. Since Reduction Rule 1 is not applicable on $\delta^+(Z)$, removing $\{a, a'\}$ from the graph reduces $\lambda(L, L')$ by at most 1 and therefore, we have that $\mu(I') < \mu$. This completes the proof of correctness of the algorithm.

Running time. We prove that on an instance $I = (D = (V, A), \sigma, \mathcal{T}, k, L)$, the algorithm computes a search tree with at most $(2\sqrt{d})^{\mu(I)}$ leaves. We have already proved that in each branch, the measure $\mu(I)$ decreases by at least 1 and since we only have $2d$-way branchings, the number of nodes of the search tree is clearly bounded by $(2d)^{\mu(I)}$. However, we analyze more closely a branch which occurs in a $2d$-way branching where $\mu(I)$ decreases by exactly 1. Suppose that J was the violating set computed in this step and suppose $v \in J$ be such that $\lambda(v, v') = 1$. Consider the branch where we recurse on the instance $(D = (V, A), \sigma, \mathcal{T}, k, \{v\})$. In this recursion, we first observe that the reduction rule will not be applied. This is because the reduction rule requires a minimum $\{v\}$-$\{v'\}$ separator of size at least 2 while $\lambda(v, v') = 1$. Therefore, the algorithm of Lemma 10.2.10 will find a $(\{v\}, k)$-component Z.

Claim 14. $|\delta^+(Z)| = 1$.

Proof. Suppose not and let $(a, b), (p, q) \in \delta^+(Z)$. Furthermore, let S be a minimum Z-Z' separator. Then, since $|S| = 1$ by our assumption, either $(a, b) \notin S$ or $(p, q) \notin S$. Suppose that $(a, b) \notin S$. Then, S is also a minimum $Z \cup \{b\}$-$Z' \cup \{b'\}$ separator, which contradicts the maximality of Z as a $(\{v\}, k)$-component. □

Consider the branching performed on the single arc in $\delta^+(Z)$. We have already shown that the measure decreases by 1 in each of these branchings. Therefore, we combine these 2 branches with the branch where we decided to recurse on the instance $(D = (V, A), \sigma, \mathcal{T}, k, \{v\})$. This leads to 2 branches where the measure decreases by 2 in each. For each branch in the $2d$-way branching where the measure decreases by 1, we can do the same to obtain at most $4d$ branches in each of which the measure decreases by 2 (see Figure 10.5). Therefore, our worst case branching is a $4d$-way branching, where the measure drops by 2 in each branch. Thus, we obtain a bound of $(2\sqrt{d})^{\mu(I)} \leq (4d)^k$ on the number of leaves of the search tree.

Since finding a violating set can be done in time $\mathcal{O}(m+n+\ell)$, computing an (L, k)-component or an irregular minimum L-L' separator can be done in time $\mathcal{O}(k^3(m + n))$ and there can be at most k applications of Reduction Rule 1 along any root to leaf path of this search tree, we have the claimed bound of $\mathcal{O}((4d)^k k^4(m+n+\ell))$ on the running time, completing the proof of Theorem 10.1.1. □

We observe here that ℓ occurs in the running time simply because the time taken to compute a violating set is $\mathcal{O}(\ell+m)$. However, in some cases, the set $\mathcal{T}$ may be given in the form of a violation oracle, in which case, the running time bound remains the same if violation oracle runs in time $\mathcal{O}(\ell)$. Therefore, the theorem below follows from the proof of Theorem 10.1.1.

Theorem 10.3.3. *There is an algorithm for d-SKEW-SYMMETRIC MULTICUT that, given a tuple $(D = (V, A), \sigma, k)$ along with a violation oracle for a family $\mathcal{T}$, runs in time $\mathcal{O}((4d)^k k^4(\ell + m + n)$ and either returns a skew-symmetric multicut of size at most $2k$ or correctly concludes that no such set exists, where ℓ is the time required for the violation oracle to compute a violated set in the family $\mathcal{T}$, $m = |A|$ and $n = |V|$.*

10.4 Applications

In this section we use the algorithm developed for d-SKEW-SYMMETRIC MULTICUT to obtain linear time parameterized algorithms for several other problems.

10.4.1 Linear time algorithm for ALMOST 2-SAT

It is known that the *variable* version of ALMOST 2-SAT, namely ALMOST 2 SAT(V) can be reduced to the *clause* version ALMOST 2-SAT via a linear time reductions [79]. Therefore, it suffices for us to give a reduction from the clause version of ALMOST 2-SAT to d-SKEW-SYMMETRIC MULTICUT. In order to give this reduction, we begin by recalling the notion of implication graphs of a 2CNF formula.

Definition 10.4.1. *Given a* 2CNF *formula F, the* **implication graph** *of F is denoted by D(F) and is defined as follows. The vertex set of the graph is the set of literals of F and for every clause $\{l_1, l_2\}$ in F, we have arcs $(\bar{l}_1, l_2)$ and $(\bar{l}_2, l_1)$.*

Clearly, implication graphs are skew-symmetric graphs where the involution σ is defined as $\sigma(l) = \bar{l}$ and $\sigma(l_1, l_2) = (\bar{l}_2, \bar{l}_1)$.

Theorem 10.4.2. [3] *A* 2CNF *formula F is satisfiable if and only if no literal and its complement are contained in the same strongly connected component of D(F).*

Observation 10.4.3. *Given a* 2CNF *formula F, let D be the implication graph of this formula and let C be a subset of the clauses of F. Let C_D be the corresponding set of arcs in the graph D. Then, the implication graph of $F - C$ is the same as the graph $D \setminus C_D$.*

Lemma 10.4.4. *Let F be a* 2CNF *formula on n variables $x_1, \ldots, x_n$. Then, (F, k) is a* YES *instance of* ALMOST 2-SAT *iff* $(D(F), \mathcal{T} = \{\{x_1\}, \ldots, \{x_n\}\}, k)$ *is a* YES *instance of* 1-SKEW-SYMMETRIC MULTICUT.

Proof. Suppose that C is a set of clauses such that $|C| \leq k$ and $F - C$ is satisfiable and let C_D be the corresponding set of arcs in D. Then, by Theorem 10.4.2, no literal of F appears in the same strongly connected component as its complement in the implication graph of $F - C$. However, by Observation 10.4.3, we have that C_D is a set of arcs such that no vertex and its conjugate lie in the same strongly connected component of $D \setminus C_D$, which implies that C_D is a solution for the instance of 1-SKEW-SYMMETRIC MULTICUT.

Conversely, let C_D be a self-conjugate set of arcs such that $|C_D| \leq 2k$ and no vertex in $\mathcal{T}$ lies in the same strongly connected component as its conjugate in $D \setminus C_D$. Let C be the set of clauses of F corresponding to C_D. Since C_D is self-conjugate, we have that $|C| \leq k$. Then, by Observation 10.4.3 and Theorem 10.4.2, the formula $F - C$ is satisfiable, which implies that C is indeed a solution for the instance of ALMOST 2-SAT. This completes the proof of the lemma. □

Since the graph $D(F)$ can be constructed in time $\mathcal{O}(|F|)$ and has $\mathcal{O}(|F|)$ arcs, we have the following theorem.

Theorem 10.4.5. *There is an algorithm that, given an instance (F, k) of* ALMOST 2-SAT *(*ALMOST 2 SAT(V)*), runs in time $\mathcal{O}(4^k k^4 \ell)$ and either returns an assignment satisfying all but at most k clauses of F or correctly concludes that no such assignment exists.*

Furthermore, there are known linear time parameter preserving reductions from EDGE BIPARTIZATION to ODD CYCLE TRANSVERSAL and from ODD CYCLE TRANSVERSAL to ALMOST 2-SAT (see [98, Page Number – 72] and [62]). The reduction from EDGE BIPARTIZATION to ODD CYCLE TRANSVERSAL increases the the number of edges and vertices in the graph by a factor of $\mathcal{O}(k)$ and the reduction from ODD CYCLE TRANSVERSAL to ALMOST 2-SAT is both parameter as well as size preserving. Therefore, have the following corollaries.

Theorem 10.4.6. EDGE BIPARTIZATION *and* ODD CYCLE TRANSVERSAL *can be solved in time $\mathcal{O}(4^k k^5(m+n))$ and $\mathcal{O}(4^k k^4(m+n))$ respectively where m and n are the number of edges and vertices in the input graph respectively.*

10.4.2 Linear time algorithm for DELETION q-Horn BACKDOOR SET DETECTION

Recall that a CNF formula F is q-Horn if there is a *certifying function* $\beta : \text{var}(F) \cup \overline{\text{var}(F)} \rightarrow \{0, \frac{1}{2}, 1\}$ with $\beta(x) = 1 - \beta(\bar{x})$ for every $x \in \text{var}(F)$ such that $\sum_{l \in C} \beta(l) \leq 1$ for every clause C of F. In this subsection, we prove Theorem 10.4.10. We begin by recalling the notion of a quadratic cover given by Boros et al. [9] which was also used in the previous chapter.

Definition 10.4.7. *Given a CNF formula F, the* **quadratic cover** *of F, is a Krom formula denoted by F_2 and is defined as follows. Let $x_1, \dots, x_n$ be the variables of F. For every clause C, we have $|C| - 1$ new variables $y_1^C, \dots, y_{|C|-1}^C$. We order the literals in each clause according to their variables, that is, a literal of x_i will occur before a literal of x_j if $i < j$. Let $l_1^C, \dots, l_{|C|}^C$ be the literals of the clause C in this order. The quadratic cover is defined as.*

$$F_2 = \bigcup_{C \in F} \bigcup_{1 \le i \le |C|-1} \{\{l_i^C, y_i^C\}, \{\bar{y}_i^C, l_{i+1}^C\}\} \ \cup \ \bigcup_{C \in F} \bigcup_{1 \le i \le |C|-2} \{\{\bar{y}_i^C, y_{i+1}^C\}\}.$$

Observation 10.4.8. *If F is a CNF formula of length ℓ, then the number of arcs in $D(F_2)$ is $\mathcal{O}(\ell)$.*

We require the following characterization of q-Horn formulas.

Lemma 10.4.9 ([9])**.** *A CNF formula F is* q-Horn *if and only if no clause of F has three literals l_1, l_2, l_3 such that each l_i and $\bar{l}_i$ are in the same strongly connected component of $D(F_2)$.*

Recall that a *deletion $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F - B \in \mathcal{C}$. These characterizations allow us to give a linear time reduction from DELETION q-Horn BACKDOOR SET DETECTION to 3-SKEW-SYMMETRIC MULTICUT.

Theorem 10.4.10. *There is an algorithm that, given an instance (F, k) of* DELETION q-Horn BACKDOOR SET DETECTION, *runs in time $\mathcal{O}(12^k k^5 \ell)$ and either returns a deletion* q-Horn*-backdoor set of F of size at most k or correctly concludes that no such set exists, where ℓ is the length of F.*

Proof. The proof is by a reduction to 3-SKEW-SYMMETRIC MULTICUT. We first construct the graph $D(F_2)$. We now define a graph D_1 which is a modification of $D(F_2)$ as follows. For every vertex l_i in $D(F_2)$ corresponding to a positive literal in F, we have two vertices l_i^+ and l_i^- and an arc (l_i^-, l_i^+) and for every vertex l_i in $D(F_2)$ corresponding to a negative literal we have two vertices $\bar{l}_i^+$ and $\bar{l}_i^-$ and an arc $(\bar{l}_i^+, \bar{l}_i^-)$. We say that an arc (l_i^-, l_i^+) *corresponds* to a (positive) literal l_i and an arc $(\bar{l}_i^+, \bar{l}_i^-)$ *corresponds* to a (negative) literal l_i.

Now, for every vertex y in $D(F_2)$ which does not correspond to a literal of F, we add vertices $y_1, \dots, y_{k+1}$ and for every arc (y, l_i) in $D(F_2)$, if l_i is a positive literal, then we add arcs $(y_1, l_i^-), \dots, (y_{2k+1}, l_i^-)$ and if l_i is a negative literal, then we add arcs $(y_1, \bar{l}_i^-), \dots, (y_{2k+1}, \bar{l}_i^-)$. For every arc (l_i, y) in $D(F_2)$, if l_i is a positive literal then we add arcs $(l_i^+, y_1), \dots, (l_i^+, y_{2k+1})$ and if l_i is a negative literal then we add arcs $(\bar{l}_i^-, y_1), \dots, (\bar{l}_i^-, y_{2k+1})$. This completes the construction of D_1. Clearly, D_1 is also skew-symmetric. The purpose of modifying the graph $D(F_2)$ is simply to map literals of the input formula to arcs in the skew-symmetric graph and conversely to ensure that arcs which do not correspond to literals of the formula F are unlikely to participate in skew-symmetric multicuts of size at most k. We note that $\{l_1^+, l_2^+\}$ are contained in the same strongly connected component of D_1 if and only if $\{l_1, l_2\}$ are in the same strongly connected component of $D(F_2)$.

We now claim that (F, k) is a YES instance of DELETION q-Horn BACKDOOR SET DETECTION if and only if $(D_1, \mathcal{T}, k, \emptyset)$ is a YES instance of 3-SKEW-SYMMETRIC MULTICUT where $\mathcal{T}$ is the set of all triples of literals $\{l_1^+, l_2^+, l_3^+\}$ in F such that l_1, l_2, l_3 occur in a clause in F.

Consider a solution S for the instance of DELETION q-Horn BACKDOOR SET DETECTION and let S_D be the set of those arcs in D_1 which correspond to the literals of the variables in S. Clearly, S_D is self-conjugate and $|S_D| \leq 2k$. We claim that S_D is a skew-symmetric multicut for the given instance. If this were not the case, then there is a clause $C \in \mathcal{C}(F)$ and literals $l_1, l_2, l_3 \in C$ such that l_1^+, l_2^+, l_3^+ each lie in the same strongly connected component of $D_1 \setminus S_D$ as their complements. Recall that by $\mathcal{C}(F)$ we denote the set of clauses of a CNF formula F.

However, by Lemma 10.4.9, there is no violating triple in the graph $D(F_2) \setminus \text{lit}(S)$ and therefore, there cannot be a violated set in the graph $D_1 \setminus S_D$.

Conversely, consider a solution S_D for the instance of 3-SKEW-SYMMETRIC MULTICUT. It is easy to see from the construction of D_1 that S_D is disjoint from arcs incident on any y_i^C. Therefore, the arcs in S_D correspond to literals and hence variables in F. Let S be the this set of variables. We claim that S is a q-Horn deletion backdoor set. If this were not the case, then by Lemma 10.4.9, there is a clause $C \in \mathcal{C}(F)$ and literals $l_1, l_2, l_3 \in C$ such that l_1, l_2, l_3 each lie in the same strongly connected component of $D(F_2) \setminus \text{lit}(S)$ as their complements. However, this implies that l_1^+, l_2^+, l_3^+ each lie in the same strongly connected component of $D_1 \setminus S_D$ as their complements, which is a contradiction. This completes the proof of correctness of the reduction.

Though this reduction is parameter preserving, it is not a linear time reduction since the number of triples we need to give as input to the 3-SKEW-SYMMETRIC MULTICUT instance could be super linear in the length of the formula. However, by using an algorithm by Boros et al. [9] that runs in time $\mathcal{O}(\ell)$ and returns a violated triple, we can use the above reduction which runs in time $\mathcal{O}(k\ell)$ and returns a skew-symmetric graph with $\mathcal{O}(k\ell)$ arcs, along with our algorithm for 3-SKEW-SYMMETRIC MULTICUT to get an algorithm which runs in time $\mathcal{O}(12^k k^5 \ell)$. This completes the proof of the theorem. $\square$

Since every deletion q-Horn backdoor set is also a strong q-Horn backdoor set, Theorem 10.4.10 has the following corollary.

Corollary 10.4.11. *There is an algorithm for* SATISFIABILITY *that runs in time* $\mathcal{O}(12^k k^5 \ell)$*, where* k *is the size of the smallest* q-Horn *deletion backdoor set of the input formula.*

10.5 Conclusion

We introduced the d-SKEW-SYMMETRIC MULTICUT problem, a general graph separation problem which generalizes a large number of well-studied problems, and described an FPT algorithm for this problem with a linear dependence on the input size and a moderate dependence on the parameter. This result gives the first linear time FPT algorithms for ODD CYCLE TRANSVERSAL, ALMOST 2-SAT and DELETION q-Horn BACKDOOR SET DETECTION. We believe that there are more graph separation problems which can be reduced to d-SKEW-SYMMETRIC MULTICUT and that our algorithm can be used as a "tool" to give (linear time) FPT algorithms for other problems which have graph separation at their core. We would like to remark that, to keep our analysis simple, we have not optimized the polynomial dependence of the running times on k.

We would also like to point out that the algorithms for variants of EDGE BIPARTIZATION and ODD CYCLE TRANSVERSAL studied in the paper of Marx et al. [76] use the almost linear time algorithm for ODD CYCLE TRANSVERSAL of Kawarabayashi and Reed or the quadratic time algorithm of Reed et al. [91]. Therefore, using our algorithm instead of these algorithms results in linear time FPT algorithms for these variants studied by Marx et al.

Part VI

Conclusion

11 Conclusion

In this thesis, the main topic of study was generalizations of the problem of finding the minimum set of vertices disconnecting two vertices in graph, also called graph separation problems. These problems not only have independent applications, motivating their study, but a variety of classical problems which are not graph separation problems or even graph problems in some cases, have been found to have a graph separation problem at their core with examples including, but not restricted to VERTEX COVER, ALMOST 2-SAT, DELETION q-Horn BACKDOOR SET DETECTION, and SATISFIABILITY. Since almost all natural graph separation problems turn out to be NP-complete, we studied these problems in the framework of Parameterized Complexity and we designed new techniques and frameworks to obtain new as well as improved FPT algorithms for certain kinds of parameterized graph separation problems. We also give applications of these techniques by showing certain problems which not graph separation problems themselves to have some variant of graph separation at their core, to give new as well as improved FPT algorithms. In particular,

- We introduced a generalization of important separators and show how to use it as a generic technique to design algorithms for problems which are not amenable to the existing machinery.

- We gave a generic subroutine which can be used to impose certain structure on a given graph which in turn leads to faster algorithms. In this thesis, we present a version of this subroutine which is very useful for graph separation problems which are parity based.
- We introduced a framework in which we can obtain greedy factor-OPT approximation algorithms for graph separation problems.

11.1 New FPT algorithms

- VERTEX COVER parameterized above the optimum value of the LP can be solved in time $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$, where n is the number of vertices in the input graph (Chapter 7).
- PARITY MULTIWAY CUT can be solved in time $\mathcal{O}(2^{\mathcal{O}(k^3)} n^{\mathcal{O}(1)})$, where n is the number of vertices in the input graph (Chapter 8).
- d-SKEW-SYMMETRIC MULTICUT can be solved in time $\mathcal{O}((4d)^k k^4(m+n))$ where m and n are the number of edges and vertices in the input graph respectively (Chapter 11).
- DELETION q-Horn BACKDOOR SET DETECTION can be solved in time $\mathcal{O}(12^k k^5 \ell)$ where ℓ is the length of the input formula (Chapter 10).
- SATISFIABILITY can be solved in time $\mathcal{O}(12^k k^5 \ell)$ where k is the size of the smallest q-Horn deletion backdoor set of the given formula and ℓ is the length of the input formula (Chapter 10).

11.2 FPT algorithms with improved dependence on the parameter

- We obtained $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$ algorithms for VERTEX COVER parameterized above the size of the maximum matching and ODD CYCLE TRANSVERSAL where n is the number of vertices in the input graph. This is the first improvement (with respect to the parameter) for ODD CYCLE TRANSVERSAL after the first algorithm of Reed et al. (2004) (Chapter 7).
- We obtained $\mathcal{O}(2.3146^k \ell^{\mathcal{O}(1)})$ algorithms for ALMOST 2-SAT where ℓ is the length of the input formula (Chapter 7).

11.3 FPT algorithms with improved dependence on the input size

- We obtained an algorithm for ODD CYCLE TRANSVERSAL which runs in time $\mathcal{O}(4^k k^4(m+n))$, which is the first linear time FPT algorithm for this problem and answers a question asked by Reed et al. (2004) (Chapter 11).
- We obtained an algorithm for ALMOST 2-SAT which runs in time $\mathcal{O}(4^k k^4 \ell)$ and an algorithm for VERTEX COVER parameterized above the size of the maximum matching running in time $\mathcal{O}(4^k k^4(m+n))$ given a graph with a matching (Chapter 11).

11.4 Future Directions

The following are potential future directions which our work in this thesis points towards.

- Consider the following parity based generalization of MULTICUT, namely PARITY MULTICUT where given a graph, integer k, vertex pairs $(s_1, t_1), \ldots, (s_r, t_r)$ and a set of parities $(p_1, \ldots, p_r)$ one for each pair, the objective is to test if there are k vertices which intersect all paths from s_i to t_i of parity p_i for every $1 \leq i \leq r$. This problem is also a clear generalization of PARITY MULTIWAY CUT. We remark that the parameterized complexity of this problem parameterized by the solution size k is not known and it could be an interesting direction to work towards and might lead to newer techniques or a much better understanding of the existing graph separation machinery.

- Recall that our algorithm for d-SKEW-SYMMETRIC MULTICUT does not have a polynomial dependence on d. It would be interesting to study the parameterized complexity of this problem for unbounded d. Since even for $d = 1$ and $d = 3$ this problem generalizes a lot of well studied problems, an FPT result for unbounded d could have significant consequences for a much larger number of problems. On the other hand, if this problem turns out to be W[1]-hard, then it could be interesting to classify the "boundary of hardness" with respect to d.

- Kratsch and Wahlström [64] studied the kernelization complexity of ABOVE GUARANTEE VERTEX COVER and obtained a randomized polynomial sized kernel for this problem through matroid based techniques. This implied a randomized polynomial kernel for all problems which have a parameter preserving polynomial time reduction to ABOVE GUARANTEE VERTEX COVER. However, for two main problems studied in this thesis, namely PMWC and d-SKEW-SYMMETRIC MULTICUT, the kernelization complexity is open. We believe that any result on the kernelization complexity of these two very general problems will help to understand and classify the structures in graph separation problems which may act as obstacles to polynomial kernels.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*, Prentice Hall Inc., 1993. Theory, algorithms, and applications. vii

[2] M. Alekhnovich and A. A. Razborov, *Satisfiability, branch-width and Tseitin tautologies*, in Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02), 2002, pp. 593–603. 205

[3] B. Aspvall, M. F. Plass, and R. E. Tarjan, *A linear-time algorithm for testing the truth of certain quantified boolean formulas*, Inf. Process. Lett., 8 (1979), pp. 121–123. 187, 238

[4] C. Berge, *Two theorems in graph theory*, in Proc. of the National Academy of Science, vol. 43, 1957, pp. 842–844. 42, 45

[5] P. Bjesse, T. Leonard, and A. Mokkedem, *Finding bugs in an alpha microprocessor using satisfiability solvers*, in CAV, 2001, pp. 454–464. 185

[6] H. L. Bodlaender, *Kernelization: New upper and lower bound techniques*, in IWPEC, J. Chen and F. V. Fomin, eds., vol. 5917 of Lecture Notes in Computer Science, Springer, 2009, pp. 17–37. 6

[7] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch, *Preprocessing for treewidth: A combinatorial analysis through kernelization*, in ICALP (1), 2011, pp. 437–448. 124

[8] E. Boros, Y. Crama, and P. L. Hammer, *Polynomial-time inference of all valid implications for horn and related formulae*, Ann. Math. Artif. Intell., 1 (1990), pp. 21–32. 187, 190, 201

[9] E. Boros, P. L. Hammer, and X. Sun, *Recognition of q-Horn formulae in linear time*, Discr. Appl. Math., 55 (1994), pp. 1–13. 187, 188, 190, 191, 193, 194, 239, 240, 242

[10] N. Bousquet, J. Daligault, and S. Thomassé, *Multicut is fpt*, in STOC, 2011, pp. 459–468. viii, 13, 135

[11] N. Bousquet, J. Daligault, S. Thomassé, and A. Yeo, *A polynomial kernel for multicut in trees*, in STACS, 2009, pp. 183–194. 6

[12] L. Cai, *Fixed-parameter tractability of graph modification problems for hereditary properties*, Information Processing Letters, 58 (1996), pp. 171–176. 80

[13] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger, *Improved algorithms for the feedback vertex set problems*, in WADS, 2007, pp. 422–433. 5

[14] J. Chen, I. A. Kanj, and W. Jia, *Vertex cover: Further observations and further improvements*, J. Algorithms, 41 (2001), pp. 280–301. 6

[15] J. Chen, I. A. Kanj, and G. Xia, *Improved upper bounds for vertex cover*, Theor. Comput. Sci., 411 (2010), pp. 3736–3756. 59, 84

[16] J. Chen, Y. Liu, and S. Lu, *An improved parameterized algorithm for the minimum node multiway cut problem*, Algorithmica, 55 (2009), pp. 1–13. 13, 22

[17] 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 (2008). viii, 5, 13

[18] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx, *Directed subset feedback vertex set is fixed-parameter tractable*, in ICALP (1), 2012, pp. 230–241. 13

[19] R. H. Chitnis, M. Hajiaghayi, and D. Marx, *Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset*, in SODA, 2012, pp. 1713–1725. 13, 135

[20] M. Chlebík and J. Chlebíková, *Crown reductions for the minimum weighted vertex cover problem*, Discr. Appl. Math., 156 (2008), pp. 292–312. 125

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, second edition*, 2001. 7

[22] Y. Crama, O. Ekin, and P. L. Hammer, *Variable and term removal from Boolean formulae*, Discr. Appl. Math., 75 (1997), pp. 217–230. 186

[23] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk, *On multiway cut parameterized above lower bounds*, in IPEC, 2011, pp. 1–12. 80

[24] ——, *Subset feedback vertex set is fixed-parameter tractable*, in ICALP (1), 2011, pp. 449–461. 214

[25] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (1994), pp. 864–894. viii

[26] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens, *An o($2^{o(k)}n^3$) fpt algorithm for the undirected feedback vertex set problem*, Theory Comput. Syst., 41 (2007), pp. 479–492. 5

[27] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999. 4, 5, 59, 202

[28] R. G. Downey, M. R. Fellows, and C. McCartin, *Parameterized approximation problems*, in Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, vol. 4169 of Lecture Notes in Computer Science, Springer Verlag, 2006, pp. 121–129. 179

[29] J. Edmonds, *Paths, trees, and flowers*, Canadian J. Math., 17 (1965), pp. 449–467. 45

[30] M. R. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. A. Rosamond, and S. Saurabh, *The complexity ecology of parameters: An illustration using bounded max leaf number*, Theory Comput. Syst., 45 (2009), pp. 822–848. 124

[31] S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta, *Planar graph bipartization in linear time*, Discrete Applied Mathematics, 156 (2008), pp. 1175–1180. 212

[32] E. Fischer, J. A. Makowsky, and E. R. Ravve, *Counting truth assignments of formulas of bounded tree-width or clique-width.*, Discr. Appl. Math., 156 (2008), pp. 511–529. 205

[33] J. Flum and M. Grohe, *Parameterized Complexity Theory*, vol. XIV of Texts in Theoretical Computer Science. An EATCS Series, Springer Verlag, Berlin, 2006. 4

[34] F. V. FOMIN, S. GASPERS, D. KRATSCH, M. LIEDLOFF, AND S. SAURABH, *Iterative compression and exact algorithms*, Theor. Comput. Sci., 411 (2010), pp. 1045–1053. 5

[35] L. R. FORD, JR. AND D. R. FULKERSON, *Maximal flow through a network*, Canadian J. Math., 8 (1956), pp. 399–404. 19, 30, 197, 221

[36] R. GANIAN, P. HLINENÝ, AND J. OBDRZÁLEK, *Better algorithms for satisfiability problems for formulas of bounded rank-width*, in IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India, K. Lodaya and M. Mahajan, eds., vol. 8 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 73–83. 205

[37] N. GARG, V. V. VAZIRANI, AND M. YANNAKAKIS, *Multiway cuts in node weighted graphs*, J. Algorithms, 50 (2004), pp. 49–61. 180

[38] S. GASPERS, S. ORDYNIAK, M. S. RAMANUJAN, S. SAURABH, AND S. SZEIDER, *Backdoors to q-horn*, in STACS, 2013, pp. 67–79. viii, 210

[39] S. GASPERS AND S. SZEIDER, *Backdoors to satisfaction*, in The Multivariate Algorithmic Revolution and Beyond, vol. 7370 of Lecture Notes in Computer Science, Springer Verlag, 2012, pp. 287–317. 187, 189

[40] J. GEELEN, B. GERARDS, B. A. REED, P. D. SEYMOUR, AND A. VETTA, *On the odd-minor variant of hadwiger's conjecture*, J. Comb. Theory, Ser. B, 99 (2009), pp. 20–29. 135

[41] A. V. GOLDBERG AND A. V. KARZANOV, *Path problems in skew-symmetric graphs*, Combinatorica, 16 (1996), pp. 353–382. 209

[42] ——, *Maximum skew-symmetric flows and matchings*, Math. Program., 100 (2004), pp. 537–568. 209, 215

[43] C. P. GOMES, H. KAUTZ, A. SABHARWAL, AND B. SELMAN, *Satisfiability solvers*, in Handbook of Knowledge Representation, vol. 3 of Foundations of Artificial Intelligence, Elsevier, 2008, pp. 89–134. 185

[44] G. GOTTLOB AND S. SZEIDER, *Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems*, Comput. J., 51 (2008), pp. 303–325. 76, 80, 120

[45] 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 (2006), pp. 1386–1396. 5

[46] 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 (2006), pp. 1386–1396. 122

[47] J. GUO, H. MOSER, AND R. NIEDERMEIER, *Iterative compression for exactly solving np-hard minimization problems*, in Algorithmics of Large and Complex Networks, 2009, pp. 65–80. 5

[48] J. GUO AND R. NIEDERMEIER, *Invitation to data reduction and problem kernelization*, SIGACT News, 38 (2007), pp. 31–45. 6

[49] F. HÜFFNER, *Algorithm engineering for optimal graph bipartization*, J. Graph Algorithms Appl., 13 (2009), pp. 77–98. 81

[50] F. HÜFFNER, C. KOMUSIEWICZ, H. MOSER, AND R. NIEDERMEIER, *Fixed-parameter algorithms for cluster vertex deletion*, Theory Comput. Syst., 47 (2010), pp. 196–217. 5

[51] F. HÜFFNER, R. NIEDERMEIER, AND S. WERNICKE, *Techniques for practical fixed-parameter algorithms*, Comput. J., 51 (2008), pp. 7–25. 5

[52] K. ICHI KAWARABAYASHI, Z. LI, AND B. A. REED, *Recognizing a totally odd k_4-subdivision, parity 2-disjoint rooted paths and a parity cycle through specified elements*, in SODA, 2010, pp. 318–328. 135

[53] K. ICHI KAWARABAYASHI AND B. A. REED, *An (almost) linear time algorithm for odd cycles transversal*, in SODA, 2010, pp. 365–378. 135, 212

[54] ——, *An (almost) linear time algorithm for odd cyles transversal*, in SODA, 2010, pp. 365–378. 135

[55] K. ICHI KAWARABAYASHI, B. A. REED, AND P. WOLLAN, *The graph minor algorithm with parity conditions*, in FOCS, 2011, pp. 27–36. 135

[56] B. M. P. JANSEN AND H. L. BODLAENDER, *Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter*, in STACS, 2011, pp. 177–188. 124

[57] B. M. P. JANSEN AND S. KRATSCH, *On polynomial kernels for structural parameterizations of odd cycle transversal*, in IPEC, 2011. 124

[58] N. KAKIMURA, K. ICHI KAWARABAYASHI, AND Y. KOBAYASHI, *Erdös-pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing*, in SODA, 2012, pp. 1726–1736. 135

[59] N. Kakimura, K. ichi Kawarabayashi, and D. Marx, *Packing cycles through prescribed vertices*, J. Comb. Theory, Ser. B, 101 (2011), pp. 378–381. 135

[60] M. Karl, *Zur allgemeinen kurventheorie*, Fund. Math., 10 (1927), pp. 96–115. vii

[61] H. A. Kautz and B. Selman, *Planning as satisfiability*, in ECAI, 1992, pp. 359–363. 185

[62] S. Khot and V. Raman, *Parameterized complexity of finding subgraphs with hereditary properties*, Theor. Comput. Sci., 289 (2002), pp. 997–1008. 60, 239

[63] S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström, *Fixed-parameter tractability of multicut in directed acyclic graphs*, in ICALP (1), 2012, pp. 581–593. 13

[64] S. Kratsch and M. Wahlström, *Representative sets and irrelevant vertices: New tools for kernelization*, in FOCS, 2012, pp. 450–459. 250

[65] M. Lampis, *A kernel of order 2 k-c log k for vertex cover*, Inf. Process. Lett., 111 (2011), pp. 1089–1091. 82, 129

[66] H. R. Lewis, *Renaming a set of clauses as a Horn set*, J. of the ACM, 25 (1978), pp. 134–135. 187

[67] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh, *Faster parameterized algorithms using linear programming*, CoRR, abs/1203.0833 (2012). 214

[68] D. LOKSHTANOV AND M. S. RAMANUJAN, *Parameterized tractability of multiway cut with parity constraints*, in ICALP (1), 2012, pp. 750–761. 13

[69] D. LOKSHTANOV, S. SAURABH, AND S. SIKDAR, *Simpler parameterized algorithm for oct*, in IWOCA, 2009, pp. 380–384. 81

[70] L. LOVÁSZ AND M. D. PLUMMER, *Matching Theory*, North Holland, Oxford, 1986. 41

[71] A. G. M. PRASAD, A. BIERE, *A survey of recent advances in SAT-based formal verification*, Software Tools for Technology Transfer, 7 (2005), pp. 156–173. 185

[72] M. MAHAJAN AND V. RAMAN, *Parameterizing above guaranteed values: Maxsat and maxcut*, J. Algorithms, 31 (1999), pp. 335–354. 60

[73] D. MARX, *Parameterized graph separation problems*, Theoret. Comput. Sci., 351 (2006), pp. 394–406. viii, 13, 14, 16, 18, 36

[74] D. MARX, *Can you beat treewidth?*, Theory of Computing, 6 (2010), pp. 85–112. 204, 205

[75] D. MARX, *Chordal deletion is fixed-parameter tractable*, Algorithmica, 57 (2010), pp. 747–768. 5

[76] D. MARX, B. O'SULLIVAN, AND I. RAZGON, *Finding small separators in linear time via treewidth reduction*, Transactions on Algorithms, To Appear. 243

[77] ——, *Treewidth reduction for constrained separation and bipartization problems*, in STACS, 2010, pp. 561–572. 19, 26

[78] D. Marx and I. Razgon, *Constant ratio fixed-parameter approximation of the edge multicut problem*, Inf. Process. Lett., 109 (2009), pp. 1161–1166. 60, 120, 210

[79] ——, *Fixed-parameter tractability of multicut parameterized by the size of the cutset*, in STOC, 2011, pp. 469–478. viii, 13, 20, 60, 135, 237

[80] K. Mehlhorn, *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, vol. 2 of Monographs in Theoretical Computer Science. An EATCS Series, Springer, 1984. 5

[81] S. Mishra, V. Raman, S. Saurabh, S. Sikdar, and C. R. Subramanian, *The complexity of könig subgraph problems and above-guarantee vertex cover*, Algorithmica, 61 (2011), pp. 857–881. 46, 60, 61, 76, 80, 81, 123, 124, 128

[82] N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh, *Lp can be a cure for parameterized problems*, in STACS, 2012, pp. 338–349. 214

[83] G. L. Nemhauser and L. E. Trotter, *Properties of vertex packing and independence system polyhedra*, Mathematical Programming, 6 (1974), pp. 48–61. 82

[84] ——, *Vertex packings: Structural properties and algorithms*, Mathematical Programming, 8 (1975), pp. 232–248. 83, 84, 102

[85] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006. 3, 7

[86] N. Nishimura, P. Ragde, and S. Szeider, *Detecting backdoor sets with respect to Horn and binary clauses*, in SAT, 2004, pp. 96–103. 186, 205

[87] J.-C. Picard and M. Queyranne, *On the integer-valued variables in the linear vertex packing problem*, Mathematical Programming, 12 (1977), pp. 97–101. 83

[88] V. Raman, M. S. Ramanujan, and S. Saurabh, *Paths, flowers and vertex cover*, in ESA, vol. 6942 of Lecture Notes in Computer Science, 2011, pp. 382–393. viii, 13, 76, 120, 214

[89] I. Razgon and B. O'Sullivan, *Almost 2-sat is fixed-parameter tractable (extended abstract)*, in ICALP(1), 2008, pp. 551–562. 213

[90] ——, *Almost 2-sat is fixed-parameter tractable.*, J. Comput. Syst. Sci., 75 (2009), pp. 435–450. viii, 5, 13, 60, 187, 205

[91] B. A. Reed, K. Smith, and A. Vetta, *Finding odd cycle transversals*, Oper. Res. Lett., 32 (2004), pp. 299–301. 5, 80, 81, 212, 243

[92] M. Samer and S. Szeider, *Algorithms for propositional model counting*, J. Discrete Algorithms, 8 (2010), pp. 50–64. 205

[93] T. J. Schaefer, *The complexity of satisfiability problems*, in Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978), ACM, 1978, pp. 216–226. 187

[94] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, J. ACM, 22 (1975), pp. 215–225. 212

[95] S. THOMASSÉ, *A quadratic kernel for feedback vertex set*, in SODA, 2009, pp. 115–119. 6

[96] W. T. TUTTE, *Antisymmetrical digraphs*, Canadian J. Math., 19 (1967), pp. 1101–1117. 209

[97] M. N. VELEV AND R. E. BRYANT, *Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors*, J. Symbolic Comput., 35 (2003), pp. 73–106. 185

[98] S. WERNICKE, *On the algorithmic tractability of single nucleotide polymorphism (snp) analysis and related problems*, Master's thesis, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, (2003). 239

[99] R. WILLIAMS, C. GOMES, AND B. SELMAN, *Backdoors to typical case complexity*, in IJCAI, 2003, pp. 1173–1178. 186

[100] G. J. WOEGINGER, *Exact algorithms for np-hard problems: A survey*, Combinatorial Optimization - Eureka, You Shrink!, LNCS, (2003), pp. 185–207. 7

[101] T. ZASLAVSKY, *Orientation of signed graphs*, European Journal of Combinatorics, 12 (1991), pp. 361–375. 209

[102] B. ZELINKA, *Analoga of Menger's theorem for polar and polarized graphs*, Czechoslovak Math. J., 26(101) (1976), pp. 352–360. 209