PARAMETERIZED ALGORITHMS AND COMPUTATIONAL LOWER BOUNDS: A STRUCTURAL APPROACH

A Dissertation

by

GE XIA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2005

Major Subject: Computer Science

PARAMETERIZED ALGORITHMS AND COMPUTATIONAL LOWER BOUNDS: A STRUCTURAL APPROACH

A Dissertation

by

GE XIA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Jianer Chen
Committee Members,	Donald K. Friesen
	Jennifer L. Welch
	Catherine H. Yan
Head of Department,	Valerie E. Taylor

August 2005

Major Subject: Computer Science

ABSTRACT

Parameterized Algorithms and Computational Lower

Bounds: A Structural Approach. (August 2005)

Ge Xia, B.Arch., Tongji University;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Jianer Chen

Many problems of practical significance are known to be NP-hard, and hence, are unlikely to be solved by polynomial-time algorithms. There are several ways to cope with the NP-hardness of a certain problem. The most popular approaches include heuristic algorithms, approximation algorithms, and randomized algorithms. Recently, parameterized computation and complexity have been receiving a lot of attention. By taking advantage of small or moderate parameter values, parameterized algorithms provide new venues for practically solving problems that are theoretically intractable.

In this dissertation, we design efficient parameterized algorithms for several well-known NP-hard problems and prove strong lower bounds for some others. In doing so, we place emphasis on the development of new techniques that take advantage of the structural properties of the problems.

We present a simple parameterized algorithm for Vertex Cover that uses polynomial space and runs in time $O(1.2738^k + kn)$. It improves both the previous $O(1.286^k + kn)$-time polynomial-space algorithm by Chen, Kanj, and Jia, and the very recent $O(1.2745^k k^4 + kn)$-time exponential-space algorithm, by Chandran and Grandoni. This algorithm stands out for both its performance and its simplicity. Essential to the design of this algorithm are several new techniques that use structural information of the underlying graph to bound the search space.

For Vertex Cover on graphs with degree bounded by three, we present a still better algorithm that runs in time $O(1.194^k + kn)$, based on an "almost-global" analysis of the search tree.

We also show that an important structural property of the underlying graphs – the graph genus – largely dictates the computational complexity of some important graph problems including Vertex Cover, Independent Set and Dominating Set.

We present a set of new techniques that allows us to prove almost tight computational lower bounds for some NP-hard problems, such as Clique, Dominating Set, Hitting Set, Set Cover, and Independent Set. The techniques are further extended to derive computational lower bounds on polynomial time approximation schemes for certain NP-hard problems. Our results illustrate a new approach to proving strong computational lower bounds for some NP-hard problems under reasonable conditions.

To my parents and my wife

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dr. Jianer Chen, who has been the one that invoked my interest in algorithms and inspired me to choose computer science as my career. In the years that followed, he has been my mentor and advisor, guiding me through my Ph.D. studies with his uncompromising scholarship, uplifting personality, and generosity. If I could possibly achieve anything now or in the future, it is indebted to him.

I am immensely grateful to Dr. Donald Friesen, Dr. Jennifer Welch, and Dr. Catherine Yan. Throughout my Ph.D. studies at Texas A&M University, I was constantly turning to them for their support and help. Dr. Friesen's wise advices were always present when I needed them. I was impressed by Dr. Welch's excellency and seriousness in both research and teaching. Dr. Yan has taught me some of the most elegant mathematical techniques and tools that I can use in my future research. They will be role models of good teachers and scholars for me to follow through my future career. I am fortunate to have all of them in my advisory committee.

I would like to express my deep appreciation to my colleagues, and in particular to Dr. Iyad Kanj. I have been collaborating with Iyad ever since the early days of my Ph.D. studies. The working relationship was always productive, especially when we sat in the same room. I learned a lot from him not only about research, but also about life. I also thank Xiuzheng Huang, Fenghui Zhang, Songjian Lu, and Jie Meng, who were members of a wonderful theory group led by Dr. Chen.

I would like to take this opportunity to thank the faculty, staff, and students in the Department of Computer Science at Texas A&M University. Their support has made my study of computer science a pleasant experience and made this work possible.

I would like to thank Dr. Robert Johnson, Dr. Mark Clayton, and all my friends in the College of Architecture at Texas A&M University. Their friendship is the best thing I had during my early years at Texas A&M University.

Finally, my deepest gratitude goes to my family. To my father, my mother, and my brother, I thank them for supporting me during the years I spent far away from them while I was pursuing my degree. To my wife, I thank her for her love and her support throughout this work.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

A. Parameterized Computation 3

B. This Dissertation 5

1. Parameterized Algorithms 5

2. Computational Lower Bounds 7

C. Preliminaries 9

II IMPROVED UPPER BOUNDS FOR VERTEX COVER 12

A. A New Approach to Improving Upper Bounds 12

B. Struction and Folding 15

C. The Algorithm 19

1. Definitions and Observations 21

2. Tuples 23

3. The Algorithm **VC** 26

D. Analysis of the Algorithm 31

E. Proof of The Main Theorem 34

III LABELED SEARCH TREE AND AMORTIZED ANALYSIS . . 59

A. Amortized Analysis on Labeled Search Trees 60

B. The Main Algorithm 62

C. Analysis of the Algorithm 72

D. An Algorithm for IS-3 103

E. Comments 104

IV PROBLEMS ON GRAPHS WITH CONSTRAINED GENUS . . 106

A. Computational Complexity and Graph Genus 106

B. Genus and Parameterized Complexity 108

1. Genus and Independent Set 108

2. Genus and Dominating Set 112

C. Genus and Subexponential Time Complexity 120

D. Genus and Approximability 128

E. Comments 134

CHAPTER Page

V COMPUTATIONAL LOWER BOUNDS VIA PARAMETERIZED COMPLEXITY . . . 135

A. A New Approach to Proving Lower Bounds . . . 135
B. Satisfiability and Weighted Satisfiability . . . 138
C. Lower Bounds on Weighted Satisfiability Problems . . . 148
D. Satisfiability Problems and the W-hierarchy . . . 156
E. Linear fpt-reductions and Lower Bounds . . . 162
F. Lower Bounds on Approximation Schemes . . . 172
G. Comments . . . 183

VI SUMMARY AND FUTURE RESEARCH . . . 184

A. Thesis Summary . . . 184
B. Future Work . . . 185
1. Improve the Upper Bounds for Independent Set . . . 185
2. Tighter Analysis of Search Tree . . . 185
3. Techniques for Designing Efficient Parameterized Algorithms . . . 186
4. Graph Genus and Computational Complexity . . . 187
5. Stronger Lower Bounds . . . 188

REFERENCES . . . 189

VITA . . . 200

LIST OF TABLES

TABLE Page

I The parameters of the operations 89

II Comparison between our results and the previous results 108

LIST OF FIGURES

FIGURE Page

1 The struction operation. 16

2 General folding. 20

3 The algorithm VC 29

4 Vertex folding 63

5 Removing an induced subgraph whose binding set has only one vertex 64

6 Removing an induced subgraph whose binding set consists of two vertices 65

7 The algorithm VC3-solver 68

8 A parameterized algorithm for Independent Set 110

9 A parameterized algorithm for Dominating Set 118

10 An algorithm solving Dominating Set 124

11 The algorithm DS-Core 165

CHAPTER I

INTRODUCTION

Many important problems that arise in real-world applications are NP-hard and according to NP-Completeness theory, are unlikely to be solved efficiently. However, this fact does not obviate the need for solving them due to their practical significance. Several ways have been proposed to cope with the NP-hardness of a certain problem. The classical approaches include heuristic algorithms [1], approximation algorithms [2], and randomized algorithms [3]. However, these approaches appear to be unsatisfactory in finding exact solutions to many important optimization problems arising in areas such as database systems, bioinformatics, and communication networks.

For example, consider the Vertex Cover problem: given a graph G, find a minimum size set of vertices in G such that every edge in G is incident on at least one vertex in the set. This problem has applications in constructing *multiple sequence alignments*, which is a fundamental problem in computational biology [4, 5]. Vertex Cover is well-known to be NP-hard [6]. Even though there exists a simple ratio-2 approximation algorithm for Minimum Vertex Cover [6], finding a polynomial-time approximation algorithm for it with a constant ratio less than 2 has been notoriously difficult. In fact, it has been proven that minimum vertex cover is not approximable to a constant ratio less than 1.36 unless P=NP, and it is conjectured that approximating it within a constant ratio less than 2 is NP-hard [7]. Therefore, approximation algorithms are not satisfactory if exact or close to optimal solutions are desired. For the minimum vertex cover problem in general, heuristic algorithms and randomized algorithms do not seem to be very helpful.

The journal model is *IEEE Transactions on Automatic Control.*

Recently, there has been considerable interest in developing improved exact algorithms for solving well-known NP-hard problems under certain constraints [8, 9]. This line of efforts was motivated by both practical and theoretical research in computational sciences.

Practically, there are certain applications that require solving NP-hard problems precisely [10] and these problems are often presented under certain constraints in practice. For example, for the previously mentioned Vertex Cover problem, the instances arising from the application of constructing multiple sequence alignments usually have the property that the size of the vertex cover is much smaller than the size of the graph. Therefore, one may ask if there exists an algorithm for solving this problem whose running time depends mainly on the size of the vertex cover rather than the size of the input. Such an algorithm is called a *fixed-parameter tractable algorithm* (or *FPT algorithm*) and study on such problems has led to a new line of research called *theory of fixed-parameter tractability* [11] which is concerned with the design of efficient parameterized algorithms for computationally difficult problems. For example, despite the difficulty faced by heuristic algorithms, approximation algorithms, and randomized algorithms in finding a minimum vertex cover of small size in a large graph, parameterized algorithms have proven to be particularly suitable for this problem. For example, the best known parameterized algorithm can decide if a graph of n vertices has a vertex cover of size at most k in time $O(1.286^k + kn)$ [12]. This algorithm appears to be quite practical for parameter values up to $k = 400$ [13].

Theoretically, this line of research may lead to a deeper understanding of the structure of NP-hard problems [14, 11, 15, 16]. For many well-known NP-hard problems such as Clique, Dominating Set, and Set Cover, finding efficient parameterized algorithms with even small parameter values has been difficult. Study on such problems has motivated the *theory of fixed-parameter intractability* [11]. Research on parameterized computation not only provides a toolkit of developing efficient algorithms for some important optimization problems, but also proves computational lower bounds for other problems. For example, by proving that the query evaluation problem belongs to the class of W[1]-hard problems, Papadimitriou and Yannakakis [17] provided strong evidence that this problem is unlikely to be solved efficiently even if the query size is small. As another example, Cesati and Trevisan [18] proved that a problem does not have an efficient polynomial-time approximation scheme unless it is fixed-parameter tractable. This essentially tells us that polynomial-time approximation schemes are unlikely to be practically useful for a problem if the problem is not fixed-parameter tractable.

A. Parameterized Computation

A *parameterized problem* Q is a decision problem consisting of instances of the form (x, k), where the integer $k \geq 0$ is called the *parameter*. For example, the parameterized Vertex Cover problem is to decide, given a pair (G, k) where G is a graph and k is a non-negative integer, whether G has a set of at most k vertices such that every edge of G is incident on at least one vertex in the set.

Certain NP-hard parameterized problems become much easier if the parameters are small. In the case of Vertex Cover, the parameter is the size of the vertex set. In particular, there is an algorithm that runs in time $O(1.286^k + kn)$ and outputs a vertex cover of size at most k [12]. It suggests that the dominant factor in the time complexity, for this particular problem, is the size of the solution set, rather than the size of the input set. Therefore, the Vertex Cover problem is practically solvable if the size of the solution set is bounded by a small constant, which is often the case for real problems.

Unfortunately, not every problem has such favorable behavior. Many other NP-hard parameterized problems remain inherently difficult with even small parameter values. For example, the Independent Set problem is not believed to be solvable in time $f(k)n^{O(1)}$, for any function f.

To capture the striking difference between these two types of problems, Downey and Fellows introduced the class of the *fixed-parameter tractable problems* denoted by FPT and the class of the *fixed-parameter intractable problems* which is contained in the various levels of *W-hierarchy* [11].

The class FPT contains problems that are solvable by parameterized algorithms in time $f(k)n^{O(1)}$, where k is given as a parameter, n is the size of the input, and f is a recursive function [1]. Vertex Cover belongs to this class, along with many well known problems such as Cutwidth [19], Treewidth [20], Longest Path [21], and so on.

The W-hierarchy $\bigcup_{t \geq 0}$ W[t] has been introduced to characterize the inherent level of intractability of parameterized problems. The 0th level of the hierarchy is the class FPT, and the ith level is denoted by $W[i]$ for $i > 0$. A parameterized complexity preserving reduction (the *fpt-reduction*) has been defined as follows. A parameterized problem Q is *fpt-reducible* to another parameterized problem Q' if there is an algorithm of running time $f(k)|x|^{O(1)}$ that on an instance (x, k) of Q produces an instance $(x', g(k))$ of Q', such that (x, k) is a yes-instance of Q if and only if $(x', g(k))$ is a yes-instance of Q', where the functions $f(k)$ and $g(k)$ depend only on k. A parameterized problem Q is $W[i]$-*hard* if every problem in $W[i]$ is fpt-reducible to Q, and is $W[i]$-*complete* if in addition Q is in $W[i]$. In particular, if any $W[i]$-hard problem is in FPT, then $W[i] =$ FPT, which, to the common belief, is very unlikely. The W[1]-hardness of a parameterized problem provides a strong evidence that the problem is not fixed-parameter tractable, or equivalently, cannot be solved in time

[1]In this dissertation, we always assume that complexity functions are "nice" with both domain and range being non-negative integers and the values of the functions and their inverses can be easily computed.

$f(k)n^{O(1)}$ for any function f. It is widely believed that the W-hierarchy does not collapse to FPT unless an important NP-complete problem – Circuit Satisfiability – is solvable in subexponential time, which is widely deemed to be unlikely (for a more complete treatment on FPT and W-hierarchy, refer to the book by Downey and Fellows [11]).

A large number of parameterized problems have been proved to be hard or complete for various levels in the W-hierarchy [11]. For example, Independent Set and Clique are complete for the class W[1], Dominating Set and Set Cover is complete for the class W[2], and t-Normalized Circuit Satisfiability is complete for the class W[t] for any $t \geq 3$.

B. This Dissertation

In the dissertation, we expand the frontiers of the research on parameterized computation in several directions. Our contribution is twofold. On one hand, we design efficient parameterized algorithms that improve the upper bounds for several well-known NP-hard problems. On the other hand, we strengthen the lower bounds for some NP-hard problems based on the framework of parameterized computation. In doing so, we place emphasis on the development of new techniques that may lead to further improvements in future research. These techniques share the common feature that they take advantage of the structural properties of the problems.

1. Parameterized Algorithms

In the first half of this dissertation, we present improved computational upper bounds for some of the most well-known graph problems including Vertex Cover, Independent Set, and Dominating Set.

In Chapter II, we present a parameterized algorithm for Vertex Cover that is simpler and more efficient than any of the previous algorithms. Vertex Cover is a canonical NP-hard problem. In recent years, significant progress has been made in developing parameterized algorithms for this problem [22, 12, 23, 24, 25, 26, 27]. Most of these algorithms are based on case by case branching according to the local structures of the graph, and hence are difficult to improve. Instead of accumulatively improving the previous algorithms, we take a different approach by emphasizing the simplicity of algorithms. We start by developing new techniques that use structural information of the underlying graph to bound the search space of the algorithm. We introduce the notion of "tuple" to capture structural information of the graph that is generated during the branching process. A similar notion was used by Robson [28] in the context of finding maximum independent set, but our approach is more systematic and general. We also introduce a generalization of the "folding" technique that was commonly used in previous results [12]. We also applied a technique called "struction" introduced by [29]. These new techniques allow us to avoid tedious case-by-case branching and present an extremely simple and uniform parameterized algorithm that uses polynomial space and runs in time $O(1.2738^k + kn)$. This algorithm stands out for both its simplicity and its performance. In fact, this algorithm not only improves the previous best polynomial-space algorithm of running time $O(1.286^k + kn)$ [12], but also surpasses the previous best *exponential-space* algorithm of running time $O(1.2745^k k^4 + kn)$ [30].

In Chapter III, we extend our study on Vertex Cover. This time, we focus our attention on the techniques of analyzing branch-and-bound algorithms. We introduce a new way to analyze the search tree of branch-and-bound algorithms that is based on global conditions instead of local constraints. The key observation behind this technique is that less efficient branches remove more edges from the graph comparing to more efficient ones. Therefore, by assigning amortized cost to the number of edges and vertices removed from the graph and keeping track of the edge-to-vertex ratio as the graph changes, we were able to balance a small number of less efficient branchings in the search tree with many efficient ones. This leads to a simple algorithm with running time $O(1.194^k + n)$ for the parameterized Vertex Cover problem on degree-3 graphs, and a simple algorithm with running time $O(1.1254^n)$ for the Maximum Independent Set problem on degree-3 graphs. Both algorithms improve the previous best algorithms for the problems.

In contrast to the Vertex Cover problem, Independent Set and Dominating Set in general are believed to be intractable for parameterized algorithms. In Chapter IV, we approached these problems by exploring an important structural property of the underlying graphs – the graph genus. Using various graph coloring and planarization techniques, we showed that graph genus largely dictates the computational complexity of these problems [31]. More precisely, we presented some exact genus thresholds that determine the parameterized complexity, the subexponential time computability, and the approximability of these problems. These results show that it is possible to design efficient parameterized algorithms for Independent Set on graphs of genus bounded by $o(n^2)$, and for Dominating Set on graphs of genus bounded by $n^{o(1)}$. Previously known efficient parameterized algorithms for these two problems only exist on planar graphs and graphs of genus bounded by $O(1)$, respectively [32, 33].

2. Computational Lower Bounds

In the second half of this dissertation, we turn our attention to proving problems computationally hard based on the framework of parameterized computation.

Due to its nature, proving strong computational lower bounds is a fundamental and difficult task in theoretical computer science. The known lower-bound results are sporadic and far from being tight. For example, consider the Clique problem which asks whether a given graph has a complete subgraph of k vertices. This problem has a trivial brute-force algorithm that enumerates all possible solutions in time $O(n^k)$. A fundamental open question is whether we can expect a significant improvement over this brute-force algorithm. For example, is there an algorithm for this problem that runs in time $O(n^{\sqrt{k}})$?

In Chapter V, we develop a set of new techniques that allowed us prove almost tight computational lower bounds based on parameterized intractability hypotheses. We proved that unless an unlikely collapse occurs in parameterized complexity theory, Clique is not solvable in time $f(k)n^{o(k)}$ for any function f, even if we restrict the parameter values to be bounded by an arbitrarily small function of n. Similar strong lower bounds on the computational complexity were also derived for other NP-hard problems including Weighted Satisfiability, Dominating Set, Hitting Set, Set Cover, and Independent Set. Essential to our techniques is an important structural property of the Circuit Satisfiability problem and its variants – their variables can be encoded or decomposed into different forms. This property allows us to design a series of reductions among different variants of the Circuit Satisfiability problem that lead to the lower bound results.

The approximation algorithms for many important problems are closely related to parameterized algorithms for these problems. We extended our lower bounds techniques to prove computational lower bounds on polynomial time approximation schemes for NP-hard optimization problems. For example, we prove that the NP-hard Distinguishing Substring Selection problem, for which a polynomial time approximation scheme has been recently developed, has no polynomial time approximation scheme that can give solutions within a factor of $(1 + \epsilon)$ of optimal in time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless an unlikely collapse occurs in parameterized complexity theory. This implies that polynomial time approximation schemes are unlikely to be practically useful for this problem, because their running time has to take the form $f(1/\epsilon)n^{O(1/\epsilon)}$ and the hidden constant in the exponent of n is usually quite large. Our results illustrate a new approach for proving almost tight computational lower bounds for some NP-hard problems under reasonable conditions.

C. Preliminaries

In the rest of this chapter, we give a concise introduction to most of the terms that will be used in the later chapters. Other terms will be introduced later in their proper setting. Most of the notations given here can be found in graph theory textbooks, such as [34].

A *graph* G is a pair (V, E), where V is a set of elements referred to as *vertices* of G and $E \subseteq V \times V$ is a binary relation on V. The elements of E are 2-element subsets of V, which are referred to as *edges* of G.

A graph G is called a *directed graph* or *digraph* if the elements of E are *ordered* pairs of vertices of G, otherwise it is called an *undirected graph.* Unless otherwise stated, the graph we consider in this dissertation are all undirected graphs.

The number of vertices of a graph G is its *order* (or *size*), written as $|G|$ (or $|V|$). The number of edges of a graph is written as $|E|$. A vertex v is *incident* with an edge e if $v \in e$. If $e = \{u, v\}$ is an edge of G, the vertices u and v are called *endpoints* (or *ends*) of e and they are considered to be *adjacent.*

Let v be a vertex of a graph G. The vertices that are adjacent to v are called its *neighbors.* The set of neighbors of v is denoted by $N_G(v)$ (or simply by $N(v)$ if the reference is clear). The set $N[v]$ denotes $N(v) \cup v$. More generally, for a set of vertices $U \subseteq V$, the neighbors in $V - U$ of vertices in U are called neighbors of U and denoted by $N(U)$. $N[U]$ denotes $N(U) \cup U$. The *degree* of v, denoted by $d_G(v)$ (or $d(v)$ if the reference is clear), is the number of edges that v is incident with, or equivalently, the number of vertices in $N(v)$. A vertex of degree 0 is *isolated*. If all vertices of G have the same degree k, then G is *k-regular*, or simply *regular*.

Let $G = (V, E)$ and $H = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then H is a *subgraph* of G. If in addition, H contains all edges in G that have both ends in V', H is an *induced subgraph* of G that is *induced* by V'. For a subgraph G' of G, denote by $G - G'$ the subgraph of G obtained by removing all vertices in G' and incident edges.

A *path* in a graph G is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k-1$. A path is *simple* if all vertices in it are distinct. Unless otherwise stated, all the paths we consider in this dissertation are simple. A *cycle* in a graph G is a path $(v_0, v_1, \ldots, v_k)$ such that $v_0 = v_n$. A graph is *acyclic* is no cycle exists in the graph.

A non-empty graph $G = (V, E)$ is called *connected* if for any two vertices $u, v \in V$, there is a path $(v_0, v_1, \ldots, v_k)$ in G such that $v_0 = u$ and $v_k = v$. A maximal connected subgraph of G is called a *connected component* (or simple *component*) of G.

A set C of vertices in G is a *vertex cover* for G if every edge in G has at least one endpoint in C. Denote by $\tau(G)$ the size of a minimum vertex cover of the graph G. A set I of vertices in G is a *independent set* if no two vertices in I are joined by an edge in E. A set D of vertices in G is a *dominating set* for G if for any vertex $u \in V - D$ there is a vertex $v \in D$ such that $(u, v) \in E$.

In the following, we give formal definitions of three parameterized problems that are the main topics of this dissertation.

> Vertex Cover: given a pair (G, k) where G is a graph and k is a non-negative integer, decide whether G has a vertex cover C of at most k vertices.

Independent Set: given a pair (G, k) where G is a graph and k is a non-negative integer, decide whether G has an independent set I of at least k vertices.

Dominating Set: given a pair (G, k) where G is a graph and k is a non-negative integer, decide whether G has a dominating set D of at most k vertices.

CHAPTER II

IMPROVED UPPER BOUNDS FOR VERTEX COVER

In this chapter, we present an $O(1.2738^k + kn)$-time polynomial-space algorithm for Vertex Cover improving both the previous $O(1.286^k + kn)$-time polynomial-space algorithm by Chen, Kanj, and Jia [12], and the very recent $O(1.2745^k k^4 + kn)$-time exponential-space algorithm, by Chandran and Grandoni [30]. Most of the previous algorithms rely on exhaustive case-by-case analysis, and an underlying conservative worst-case-scenario assumption. The contribution of this algorithm lies in its *extreme* simplicity, uniformity, and obliviousness of the algorithm presented. Several new techniques are introduced to take advantage of the rich structural properties of the problem, including: *general folding*, *struction*, *tuples.* The algorithm also induces improvement on the upper bound for the Independent Set problem on graphs of degree bounded by 6.

A. A New Approach to Improving Upper Bounds

Deriving upper bounds for NP-hard problems is important from both the practical and theoretical perspectives. Practically, an algorithm of running time $O(1.01^n)$ (n is the input size) could render an NP-hard problem computational feasible for most practical instances (say for $n \leq 1000$) as opposed to an $O(2^n)$ algorithm for the problem. Theoretically, deriving upper bounds for an NP-hard problem helps studying the inherent structural complexity of the problem which can lead to a deeper understanding of the problem itself, and in general, of the structure of NP-hard problems. As a result, the study of exact algorithms for NP-hard problems has been attracting a lot of attention recently [8, 9, 35]. In particular, for many well-known NP-hard problems with important applications such as Satisfiability, Independent Set, Vertex Cover, and Graph Coloring, exact algorithms have been extensively studied and developed.

This chapter focuses on the parameterized Vertex Cover problem, abbreviated VC henceforth. There are ample reasons to start our exposition with this problem. Vertex Cover was amongst the first few problems that were shown to be NP-hard [6]. In addition, the problem has been a central problem in the study of parameterized algorithms [11], and has applications in areas such as computational biochemistry and biology [13]. Since the development of the first parameterized algorithm for the problem by Sam Buss which runs in $O(2^k k^{2k+2} + kn)$ time [36], there has been an impressive list of improved algorithms for the problem [22, 12, 23, 24, 25, 26, 27]. The most recent algorithm for the problem running in polynomial space, was presented in 1999 and gives the best time upper bound of $O(1.286^k + kn)$ [12]. Algorithms using exponential space for the problem have also been proposed [30, 12, 26], amongst which the best runs in time $O(1.2745^k k^4 + kn)$ [30]. Most of the previous algorithms rely on exhaustive case-by-case analysis, and work under a conservative worst-case-scenario assumption. The analysis of these algorithms would consider the worst-case branch over numerous combinatorial cases, and derive an upper bound accordingly. In particular, the design phase of these algorithms (usually) did not provide the appropriate tools that the analysis phase could take advantage of to derive better upper bounds than the ones claimed. Consequently, to improve the upper bounds, larger and larger sets of local structures had to be examined and processed differently. Examining these numerous structures and processing them differently on a case-by-case basis became very meticulous, rendering the verification and implementation of these algorithms very complicated and unpractical.

On the other hand, progress has been recently made on deriving computational lower bounds for the problem. It has been shown that unless all problems in the complexity class SNP are solvable in sub-exponential time, there is a constant $c_0 > 1$ such that Vertex Cover cannot be solved in time $c_0^k n^{O(1)}$ [37, 38]. Therefore, from both the algorithmic and the complexity points of view, it becomes important to study how far we can push to lower the constant $c > 1$, such that the VC problem can be solved in time $c^k n^{O(1)}$.

In this chapter we adopt a different approach to improve the time upper bound for the VC problem. Our goal was to design an algorithm that is simple and uniform, and that provides the tools and the ground for an insightful analysis of its running time. We came up with an algorithm that is extremely simple. The algorithm keeps a list of prioritized "advantageous" structures. At each stage it will pick the structure of highest priority (most advantageous structure). Picking such a structure can be easily done following few simple rules. When this structure is picked, the algorithm processes this structure very *uniformly*, and *obliviously*, in a way that is almost independent of what the structure is. As a matter of fact, there are *only* two different ways for processing *any* structure – that is, only two different branches – that the algorithm needs to distinguish. All the other operations performed by the algorithm are non-branching operations that process certain simple structures in the graph such as degree-1 and degree-2 vertices, and that set the stage for the subsequent branch performed by the algorithm to be efficient. The interleaving and ordering of these operations in the algorithm is crucial, and is fully exploited by the analysis phase.

To be able to carry out all the above, a set of new techniques and generalization of some well-known and classical techniques have been introduced. A graph operation that is a generalization of the *folding* operation [12], and a graph operation that is a specialization of the *struction* operation [29], have been developed. These operations help the algorithm remove several simple structures from the graph without the need to perform any branching. This makes analyzing the two branching operations performed in the resulting graph more insightful. The notion of a *tuple*, which was implicitly used by Robson [28], has been fully developed and exploited to prune the search space. Finally we perform a "local" amortized analysis to balance expensive branching operations by combining them with more efficient operations. Being able to perform this local amortized analysis is indebted to the careful interleaving and ordering of the operations in the algorithm, and not to the different way of processing each structure.

The presented algorithm runs in polynomial space, and has its running time bounded by $O(1.2738^k + kn)$. This is a significant improvement over the previous polynomial-space algorithm for the problem which runs in $O(1.286^k + kn)$ time. This also improves the exponential space $O(1.2745^k k^4 + kn)$-time algorithm by Chandran and Grandoni [30]. As a by-product of this algorithm, we obtain a polynomial-space $O(1.224^n)$-time algorithm for the Independent Set problem on graphs of degree bounded by 6, improving the previous best polynomial-space algorithm of running time $O(1.227^n)$ by Robson [28] on such graphs.

B. Struction and Folding

Recall that $\tau(G)$ denotes the size of a minimum vertex cover of G. The following proposition from [12] is based on a theorem by Nemhauser and Trotter [39].

Proposition II.1 ([12]) *There is an algorithm of running time $O(kn + k^3)$ that, given an instance (G, k) of the VC problem where $|G| = n$, constructs another instance (G_1, k_1) of VC with $k_1 \leq k$ and $|G_1| \leq 2k_1$, such that $\tau(G) \leq k$ if and only if $\tau(G_1) \leq k_1$.*

We say that the instance (G_1, k_1) is the *kernel* of the instance (G, k). Proposition II.1 allows us to assume, without loss of generality, that in an instance (G, k) of the VC problem the graph G contains at most $2k$ vertices.

For two vertices u and v we say that $\{u, v\}$ is an *anti-edge* in G if (u, v) is not an edge in G. Let v_0 be a vertex in G with a set of neighbors $\{v_1, \cdots, v_p\}$. Construct a graph G' as follows: (1) remove the vertices $\{v_0, v_1, \cdots, v_p\}$ from G and introduce a new node v_{ij} for every anti-edge $\{v_i, v_j\}$ in G where $0 < i < j \leq p$; (2) add an edge (v_{ir}, v_{is}) if $i = j$ and (v_r, v_s) is an edge in G; (3) if $i \neq j$ add an edge (v_{ir}, v_{js}); and (4) for every $u \notin \{v_0, \cdots, v_p\}$, add the edge (v_{ij}, u) if (v_i, u) or (v_j, u) is an edge in G. This completes the construction of G'. We say that the graph G' is obtained from G by applying the *struction* operation to the vertex v_0 in G [29] (see Figure 1 for an illustration). We have the following lemma.

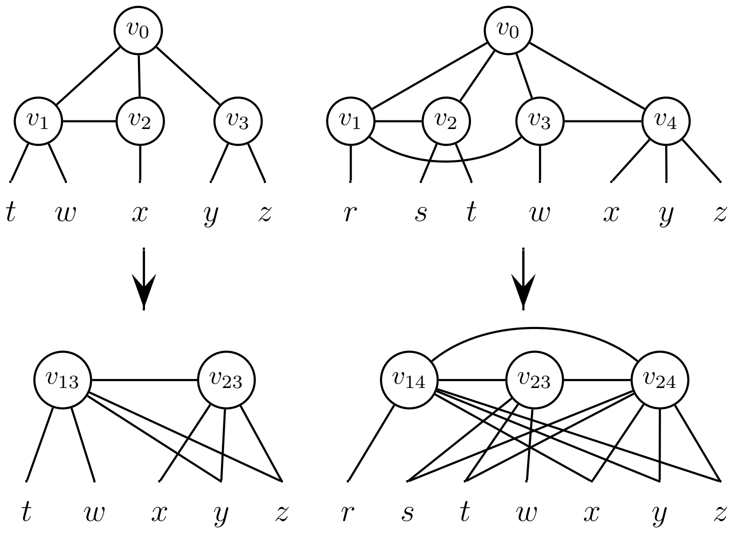

Fig. 1. The struction operation.

Lemma II.2 *Let v_0 be a vertex in G with a set of neighbors $\{v_1, \cdots, v_p\}$. Suppose that there are at most $p - 1$ anti-edges among the vertices $\{v_1, \cdots, v_p\}$, and let G' be the graph obtained from G by applying the struction operation to the vertex v_0. Then $\tau(G') \leq \tau(G) - 1$.*

Proof. Let $\alpha(G)$ and $\alpha(G')$ denote the size of a maximum independent set in G and G', respectively. It was shown in [29] that $\alpha(G') = \alpha(G) - 1$. Let n and n' denote the number of vertices in G and G', respectively. Since there are at most $p-1$ anti-edges among the vertices $\{v_1, \cdots, v_p\}$, the number of newly introduced vertices in G' is at most $p-1$. Since $p+1$ vertices were removed from G, namely $\{v_0, v_1, \cdots, v_p\}$, we have $n' \leq n-2$. It is well-known [6] that for any graph H we have $\alpha(H)+\tau(H) = |H|$. Therefore $\tau(G') = n' - \alpha(G') \leq (n-2) - (\alpha(G)-1) = \tau(G) - 1$. This completes the proof. □

Lemma II.2 gives a generic setting in which the application of the struction operation reduces the size of the minimum vertex cover of the graph. This operation turns out to be very useful in the algorithm presented in this chapter. Two possible scenarios in which the operation will be applied are illustrated in Figure 1. We will assume that we have a subroutine called **Struction()** that applies the struction operation to a vertex v in G. Note that the time spent by this operation on a vertex v is proportional to $|N(v)|$.

Remark II.3 *When the struction operation is applied to a degree-3 vertex u in G with neighbors v, w, and z, and with an edge between v and w, the only vertices removed from G are u, v, w, and z, and the only vertices of G in the resulting graph whose degree could have increased are the neighbors of z.*

Next we present an operation that generalizes the folding operation introduced in [12].

Lemma II.4 *Let I be an independent set in G and let $N(I)$ be the set of neighbors of I. Suppose that $|N(I)| = |I| + 1$, and that for every $S \subset I$, $S \neq \emptyset$, we have $|N(S)| \geq |S| + 1$.*

1. *If the graph induced by $N(I)$ is not an independent set, then there exists a minimum vertex cover in G that includes $N(I)$ and excludes I.*
2. *If the graph induced by $N(I)$ is an independent set, let G' be the graph obtained from G by removing $I \cup N(I)$ and adding a vertex u_I, then connecting u_I to every vertex $v \in G'$ such that v was a neighbor of a vertex $u \in N(I)$ in G. That is, add an edge (u_I, v) for every $v \in N(N(I)) - I$ to G'. Then $\tau(G') = \tau(G) - |I|$.*

Proof. We first prove the following claim: There exists a minimum vertex cover C for G such that C contains I and excludes $N(I)$, or such that C contains $N(I)$ and excludes I. To see why this is true, suppose that $C \cap I = X \neq \emptyset$ and $C \cap N(I) = Y \neq \emptyset$. Since C is a vertex cover for G, we have $N(I - X) \subseteq Y$. If $(I - X) \neq \emptyset$, $|Y| \geq |N(I-X)| \geq |I-X|+1 = |I|-|X|+1$, from the statement of the lemma. If $I-X = \emptyset$, since $Y \neq \emptyset$, we also have $|Y| \geq |I| - |X| + 1$. Therefore $|Y| + |X| \geq |I| + 1 = |N(I)|$. Since I is an independent set, if we replace $Y \cup X$ by $N(I)$ in C we get a vertex cover C' for G of size not larger than that of C. It follows that C' is a minimum vertex cover for G that includes $N(I)$ and excludes I, and the claim follows.

Let C be a minimum vertex cover that satisfies the conditions in the claim. If the graph induced by $N(I)$ is not an independent set, then any vertex cover of G, and in particular C, cannot exclude $N(I)$. It follows from the above claim that C a minimum vertex cover for G that includes $N(I)$ and excludes I. This proves part (1) in the statement of the lemma.

Suppose now that $N(I)$ is an independent set. If C contains I, then C excludes $N(I)$ and must include $N(N(I))$ in G'. Then $C' = C - I$ is a vertex cover for G' of size $|C| - |I| = \tau(G) - |I|$, and $\tau(G') \leq \tau(G) - |I|$. If C contains $N(I)$, then $(C - N(I)) \cup \{u_I\}$ is a vertex cover for G' of size $\tau(G) - (|I| + 1) + 1 = \tau(G) - |I|$, and $\tau(G') \leq \tau(G) - |I|$. This shows that $\tau(G') \leq \tau(G) - |I|$.

On the other hand, let C' be a minimum vertex cover for G'. Then either C' contains u_I or contains $N(u_I)$ and excludes u_I. If C' contains u_I, then $(C' - \{u_I\}) \cup N(I)$ is a vertex cover for G os size $|C'| + |I|$, and $\tau(G') \geq \tau(G) - |I|$. If C' contains $N(u_I)$ and excludes u_I, then $C' \cup I$ is a vertex cover for G of size $|C'| + |I|$, and $\tau(G') \geq \tau(G) - |I|$. This shows that $\tau(G') \geq \tau(G) - |I|$.

It follows from the above that $\tau(G') = \tau(G) - |I|$. This proves part (2) in the statement of the lemma, and the proof is complete. □

The following proposition can be proved using the results in [40, 41].

Proposition II.5 *Let (G, k) be an instance of* **VC**. *If a structure to which Lemma II.4 applies exists in G, then such a structure can be found in $O(k^2\sqrt{k})$ time, otherwise, the number of vertices in G is at most $2k$.*

We will refer to the operation in Lemma II.4 by the *general folding* operation. The reason behind this nomenclature is that this operation generalizes the folding operation that appeared in [12, 42], and which deals with the case when $|I| = 1$. Two scenarios in which this operation is applicable are given in Figure 2. Scenario (a) is the special case in which the general folding reduces to the folding operation. We will assume that we have a subroutine called **General-Fold()** that searches for a structure in the graph to which the general folding operation applies, and applies the operation to it in case it exists. Using Proposition II.5, this subroutine can be implemented to run in $O(k^2\sqrt{k})$ time.

C. The Algorithm

The main algorithm is a branch-and-search process. Each stage of the algorithm starts with an instance (G, k) of VC, and tries to reduce the parameter k by identifying a set S of vertices that are entirely contained in a minimum vertex cover of G, and

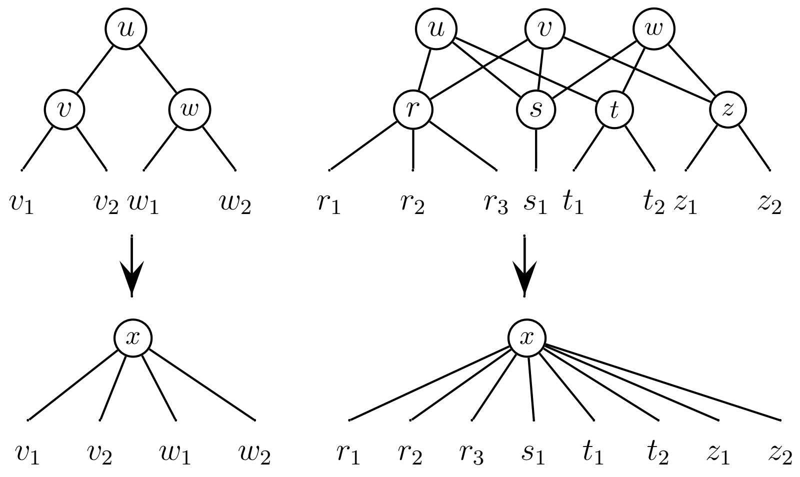

Fig. 2. General folding.

including the vertex set S in the objective minimum vertex cover, which will be called *the partial cover* (or simply the cover) for G, then recursively works on the reduced instances. The subroutine **General-Fold**(G) applies the general folding operation to G. Similarly, the subroutines **Struction**(G) and **Kernelize**(G) apply the struction operation and the kernelization procedure to G.

If a vertex set S is identified such that either there is a minimum vertex cover containing the entire S or there is a minimum vertex cover containing no vertex in S, then we can *branch on the set S*. This means that the algorithm constructs two instances of the VC problem, one by including the set S in the partial cover and the other by excluding the set S from the partial cover, and in the latter case, every vertex that is adjacent to a vertex in S should be included in the partial cover. The algorithm then recursively works on the two reduced instances. If the set S consists of a single vertex v, then we simply say we *branch on v*.

1. Definitions and Observations

Observation II.6 *Let v be a vertex in G. 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 C for G contains $|N(v)| - 1$ vertices from $N(v)$, then it has to contain v. We form another minimum vertex cover for G by replacing v in C by the single vertex in $N(v) - C$. We obtain a minimum vertex cover for G containing $N(v)$. $\square$

Observation II.7 *Let u and v be two adjacent vertices in G. Then there exists a minimum vertex cover for G that includes v or that excludes v and excludes at least another neighbor of u.*

Proof. Proceed by contradiction. Suppose that every minimum vertex cover C excludes v and does not exclude any other neighbor of u. Since C excludes v, C must contain u. Since C contains all the neighbors of u except v, $(C - \{u\}) \cup \{v\}$ is a minimum vertex cover for G containing v, a contradiction. $\square$

A vertex v is said to *dominate* a vertex u if (u, v) is an edge in G and $N(u) \subseteq N[v]$. A vertex u is said to be *almost-dominated* by a vertex v if u and v are non-adjacent and $|N(u) - N(v)| \leq 1$.

Observation II.8 *Let u and v be two vertices in G such that v dominates u. Then there exists a minimum vertex cover of G containing v.*

Proof. Let C be a minimum vertex cover. If C does not contain v then C must contain $N(v)$ which includes u (since (u, v) is an edge). Since $(N(u) - \{v\}) \subseteq N(v)$, if we remove u from C and replace it with v, we get a minimum vertex cover for G containing v. This completes the proof. $\square$

A *good pair* of vertices is a pair of vertices $\{u, z\}$ chosen as follows. For a vertex u in G with neighbors $\{u_1, \cdots, u_d\}$, define its *tag*, denoted $tag(u)$, to be the vector $\eta = \langle \eta_1, \cdots, \eta_d \rangle$, where η_1 is the degree of the largest-degree neighbor of u, η_2 is the degree of the second-largest degree neighbor of u, ..., and η_d is the degree of the smallest-degree neighbor of u. First choose a vertex u of minimum degree in G such that the following conditions are satisfied in their respective order.

(i) The vector $tag(u)$ is maximum in lexicographic ordering over $tag(w)$ for every w in G with the same degree as u.

(ii) If G is regular, then the number of pairs of vertices $\{x, y\} \subseteq N(u)$ such that y is almost dominated by x is maximized.

(iii) The number of edges in the subgraph induced by $N(u)$ is maximized.

Now choose a neighbor z of u such that the following conditions are satisfied.

(a) If there exist two neighbors of u, say v and w, such that v is almost-dominated by w, then z is almost-dominated by a neighbor of u.

(b) The degree of z is maximum among all neighbors of u satisfying part (a) above. (Note that if no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$, then (a) is vacuously satisfied by every vertex in $N(u)$, and z will be a neighbor of u of maximum degree).

(c) The degree of z in the subgraph induced by $N(u)$ is minimum among all vertices satisfying (a) and (b) above. (That is, z is adjacent to the least number of neighbors of u.)

(d) The number of shared neighbors between z and a neighbor of u is maximized over all neighbors of u satisfying (a), (b), and (c) above.

2. Tuples

Tuples will play a crucial role in the algorithm by helping to reduce the search space. We define the notion of tuples next and describe how they will be updated and processed by the algorithm.

Definition and intuition

A *tuple* is a pair (S, q) where S is a set of vertices and q is an integer. The tuple will represent the information that in the instance of the problem (G, k) we can look for a minimum vertex cover for G excluding at least q vertices from S. This information will help the algorithm prune the search tree. The algorithm will only consider tuples (S, q) with $q \leq 2$, so we will only focus on such tuples here. A tuple (S, q), where $S = \{u, v\}$, is called a *2-tuple* if it satisfies the following conditions: (1) $q = 1$, (2) $d(u) \geq d(v) \geq 1$, and (3) u and v are non-adjacent. A 2-tuple $(\{u, v\}, 1)$ is a *strong-2-tuple* if it satisfies the additional condition: $d(u) \geq 4$ and $d(v) \geq 4$, or $2 \leq d(u) \leq 3$ and $2 \leq d(v) \leq 3$.

To see how tuples can be used to prune the search space, suppose that the algorithm branches on a vertex z with neighbors $N(z)$. By Observation II.6, either there exists a minimum vertex cover in G that contains $N(z)$, or there exists a minimum vertex cover for G that excludes at least two vertices from $N(z)$. Therefore, when the algorithm branches on z, on the side of the branch where z is included, we can restrict our search to a minimum vertex cover that excludes at least two neighbors of $N(z)$, and we know that this is safe because if such a minimum vertex cover does not exist, then on the other side of the branch where $N(z)$ has been included the algorithm will still be able to find a minimum vertex cover. Consequently, on the side of the branch where z is included, we can work under the assumption that at least two vertices in $N(z)$ must be excluded. This working assumption will be stipulated by creating the tuple $(N(z), q = 2)$. This information will be used by the algorithm to prune the search space and render the branching more efficient. Similarly, if the algorithm branches on a vertex z with a neighbor u, by Observation II.7, either there exists a minimum vertex cover in G that includes z, or there exists a minimum vertex cover in G that excludes z, and excludes at least another neighbor of u. Therefore on the side of the branch where z is excluded, we can restrict our search to a minimum vertex cover that excludes at least two vertices in $N(u)$ (z and another vertex in $N(u)$). This working assumption can be stipulated by creating the tuple $(N(u), q = 2)$.

Updating tuples

Let (S, q) be a tuple. If $q = 0$ then the tuple S will be removed because the information represented by (S, q) is satisfied by any minimum vertex cover. If one of the vertices in S is removed by excluding it from the cover, then the tuple is modified by removing the vertex from S and decrementing q by 1. The correctness of this step can be seen as follows. Suppose a vertex $u \in S$ has been excluded from the cover. If there exists a minimum vertex cover C that excludes at least q vertices from S, then C excludes at least $q-1$ vertices from $S - \{u\}$. Now if a vertex $u \in S$ is removed from the graph by including it in the cover, the vertex is removed from S and q is kept unchanged. The justification of this step follows from the argument that if there exists a minimum vertex cover C that includes u and excludes at least q vertices from S, then C must exclude q vertices from $S - \{u\}$ (note that the validity of the inclusion of u in the cover is taken care of by the correctness of the steps performed by the algorithm when it includes u in the cover). If a vertex in $u \in S$ is removed from the graph as a result of applying the struction operation or the general folding operation, then u is removed from S and q is decremented by 1. The reason is that if there exists a minimum vertex cover that excludes at least q vertices from S, then this vertex cover will exclude at least $q - 1$ vertices from $S - \{u\}$.

The tuples need to be updated as described above after each operation of the algorithm. We will assume that this step is performed implicitly by the algorithm after each operation.

Branching on 2-tuples

When the algorithm creates tuples it will use them to generate 2-tuples using very simple rules described in the algorithm (steps a.2 and a.3 of the subroutine **Reducing** in Figure 3). The algorithm only processes 2-tuples of the form $(S, 1)$. A 2-tuple of the form $(\{u, z\}, 1)$ stipulates that at least one vertex in $\{u, z\}$ must be excluded from the cover. This means that if u is included in the cover then z should be excluded, and hence $N(z)$ must be included, and similarly, if z is included in the cover then u should be excluded, and $N(u)$ must be included. Let $(S = \{u, z\}, 1)$ be a 2-tuple. The algorithm will branch on a vertex in this two tuple. This vertex is picked as follows. If there is a vertex $w \in \{u, z\}$ such that w has a neighbor u' and $|N(u') - N(S - \{w\})| \leq 1$, then the algorithm will branch on the vertex in $S - \{w\}$ (that is, if there is a vertex in S with a neighbor that is almost-dominated by the other vertex in S, then the algorithm will pick the other vertex in S). Otherwise, it will pick a vertex in S arbitrarily and branch on it. Without loss of generality, we will always assume that the vertex in the 2-tuple that the algorithm branches on is z. The algorithm can be made anonymous to this choice by ordering the vertices in a 2-tuple whenever it is created.

3. The Algorithm **VC**

A tuple, a good pair, or a vertex of degree at least seven, will be referred to by the word *structure*. The algorithm will maintain a set of structures $\mathcal{T}$, and then it will pick a structure and processes it. The structures in $\mathcal{T}$ will be considered in a certain (sorted) order according to their priorities. The higher the priority of a structure is, the higher the expected benefit out of this structure will be. The priority is assigned to a structure whenever this structure is created. If an operation in the algorithm affects a certain structure in $\mathcal{T}$, then the priority of this structure needs to be modified accordingly, and the structure may need to be removed from $\mathcal{T}$. If a structure Γ is a vertex, and if this vertex is removed by the algorithm, then Γ is also removed from $\mathcal{T}$. If Γ is a good pair, and if one of the vertices in Γ is removed by the algorithm, then Γ is removed from $\mathcal{T}$. If Γ is a tuple (S, q) then Γ will be updated as described above. We will assume that the algorithm implicitly updates the structures in $\mathcal{T}$ and their priorities after each operation. We give below a list of the structures Γ that can exist at a certain point in $\mathcal{T}$ listed in a non-increasing order of their priorities. Besides the structures listed below, $\mathcal{T}$ will contain tuples that are not 2-tuples, and those tuples will not be given any priorities. The algorithm will never process these tuples, and they are only used as intermediate structures which can result in the creation of 2-tuples by the algorithm.

1 Γ is a strong 2-tuple.

2 Γ is a 2-tuple.

3 Γ is a good pair (u, z). where $d(u) = 3$ and the neighbors of u are degree-5 vertices such that no two of them share any common neighbors besides u.

4 Γ is a good pair (u, z) where $d(u) = 3$, $d(z) \geq 5$.

5 Γ is a good pair (u, z) where $d(u) = 3$, $d(z) \geq 4$.

6 Γ is a good pair (u, z) where $d(u) = 4$, u has at least three degree-5 neighbors, and the graph induced by $N(u)$ contains at least one edge, i.e., there is at least one edge among the neighbors of u.

7 Γ is a good pair (u, z) where $d(u) = 4$ and all the neighbors of u are degree-5 vertices such that no two of them share a neighbor other than u.

8 Γ is a vertex z with $d(z) \geq 8$.

9 Γ is a good pair (u, z) where $d(u) = 4$, $d(z) \geq 5$.

10 Γ is a good pair (u, z) where $d(u) = 5$, $d(z) \geq 6$.

11 Γ is a vertex z such that $d(z) \geq 7$.

12 Γ is any good pair other than the ones appearing in 1–11 above.

We note that the above list gives the structures that could exist in $\mathcal{T}$ and their priorities. Moreover, the above list is exhaustive in the sense that for any non-empty graph G, G must contain one of the structures listed above, and the algorithm will have a structure to process. This can be seen as follows. First if the degree of G is bounded by 2 then **Reducing** must apply. So suppose that this is not the case. Suppose also that G is connected [1]. If G contains a vertex of degree at least 7, then the algorithm will have at least one structure to consider by item 11 on the list. If this is not the case, then G has degree bounded by 6. If G is regular, then any good pair (u, z) must satisfy $d(u) = d(z)$, and hence none of the items 3-7, 9-10, dealing with good pairs applies, and item 12 applies. Basically, item 12 deals with regular

[1]If G is disconnected, the algorithm will be called recursively on each connected component of G (see Theorem V.10).

graphs. Suppose now that G is not regular. Let u be a vertex with minimum degree in G. If $d(u) = 3$ then item 5 applies (since G is not regular). If $d(u) = 4$ then item 9 applies. If $d(u) = 5$ then item 10 applies. Note that since G is not regular and has degree bounded by 6, G must contain a vertex of degree bounded by 5. This shows that the above list is comprehensive.

The algorithm will return the size of the minimum vertex cover in case this size is bounded by k, or otherwise it will reject. The algorithm can be easily modified to return the desired minimum vertex cover itself in case it has size bounded by k. We present the algorithm and prove its correctness next, and we analyze its running time in the next section. The algorithm is given in Figure 3. Note that the algorithm performs *only* two branches regardless of the structure picked, which are the ones given in step 3 of the algorithm.

Proposition II.9 *The operations in step a of* **Reducing** *are valid tuple operations.*

PROOF. If $|S| < q$ then the information represented by the tuple (S, q) has been violated because there does not exist a minimum vertex cover that excludes q vertices from S, and hence the algorithm can reject the instance. This shows that step a.1 is valid. (Again we note here that it is "responsibility" of the algorithm to guarantee that whenever it branches by creating a tuple on one side of the branch, then either there exists a minimum vertex cover that does not violate the tuples along this side of the branch, or there is a minimum vertex cover along the other side of the branch.) If (S, q) is a tuple and $u \in S$, and if there exists a minimum vertex cover excluding q vertices from S, then there exists a minimum vertex cover excluding $q - 1$ vertices from $S - \{u\}$. Therefore step a.2. is correct. Now let us look at step a.3 in **Reducing**. Suppose (S, q) is tuple such that there are two vertices u and v in S that are adjacent. If there exists a minimum vertex cover C for G that excludes at least q vertices from

```
$\mathbf{VC}(G, \mathcal{T}, k)$
  Input: a graph $G$, a set $\mathcal{T}$ of tuples, and a positive integer $k$.
  Output: the size of a minimum vertex cover of $G$ if the size is bounded by $k$;
    report failure otherwise.

  0. if $|G| > 0$ and $k = 0$ then reject;
  1. apply Reducing;
  2. pick a structure $\Gamma$ of highest priority;
  3. if ($\Gamma$ is a 2-tuple $(\{u, z\}, q)$) or ($\Gamma$ is a good pair $(u, z)$ such that $z$ is almost-
    dominated by a vertex $v \in N(u)$) or ($\Gamma$ is a vertex $z$ with $d(z) \geq 7$) then
          return $\min\{1+\mathbf{VC}(G - z, \mathcal{T} \cup (N(z), 2), k - 1),$
                 $d(z)+ \mathbf{VC}(G - N[z], \mathcal{T}, k - d(z))\}$;
      else if $\Gamma$ is a good pair $(u, z)$ then
          return $\min\{1+\mathbf{VC}(G - z, \mathcal{T}, k - 1),$
                 $d(z)+ \mathbf{VC}(G - N[z], \mathcal{T} \cup (N(u), 2), k - d(z))\}$;

Reducing
  a. for each tuple $(S, q) \in \mathcal{T}$ with $q = 2$ do
      a.1. if $|S| < q$ then reject;
      a.2. for every vertex $u \in S$ do $\mathcal{T} = \mathcal{T} \cup \{(S - \{u\}, q - 1)\}$;
      a.3. if $S$ is not an independent set then
          $\mathcal{T} = \mathcal{T} \cup (\bigcup_{(u,v) \in E, u,v \in S} \{(S - \{u, v\}, q - 1)\})$;
      a.4. if there exists $v \in G$ such that $|N(v) \cap S| \geq |S| - q + 1$ then
          return $(1+\mathbf{VC}(G - v, \mathcal{T}, k - 1))$; exit;
  b. if there exists $v \in G$ such that $d(v) = 1$ then
      return $(1+ \mathbf{VC}(G - N[v], \mathcal{T}, k - 1))$; exit;
  c. if General-Fold$(G)$ or Conditional_Struction$(G)$ in the given
    order is applicable then apply it; exit;
  d. if there are $u$ and $v$ in $G$ such that $v$ dominates $u$ then
      return $(1+ \mathbf{VC}(G - v, \mathcal{T}, k - 1))$; exit;

Conditional_Struction
  if there exists a strong 2-tuple $\{u, v\}$ in $\mathcal{T}$ then
      if there exists $w \in \{u, v\}$ such that $d(w) = 3$ and the
      Struction is applicable to $w$ then apply it;
  else if there exists a vertex $u \in G$ such that the Struction is applicable to $u$
      then apply it;
```

Fig. 3. The algorithm VC

S, then C must exclude at least $q-1$ vertices from $S-\{u,v\}$ since C must contain at least one of the vertices in $\{u,v\}$. Therefore $(S-\{u,v\}, q-1)$ is a tuple, and the statement is correct. Now let us look at step a.4. Again since (S,q) is a tuple, there exists a minimum vertex cover C that excludes at least q vertices from S. Since $|N(v)\cap S| \geq |S|-q+1$, C excludes at least one vertex in $N(v)$ and must include v. Therefore there exists a minimum vertex cover of G that includes v and the statement is correct. □

Theorem II.10 *The algorithm* **VC** *is correct.*

Proof. We look at the operations performed by the algorithm. Step a of **Reducing** is valid by Proposition II.9. Step b in **Reducing** is correct because if $d(v)=1$ then there exists a minimum vertex cover excluding v and including the neighbor of v. Therefore G has a vertex cover of size k if and only if $G-N[v]$ has a vertex cover of size $k-1$. By Lemma II.2, the struction operation is correct and hence the operation **Conditional_Struction** is correct as well, since it only applies the struction operation to certain vertices that meet some specified conditions. The same is also true for the operation **General-Fold** by Lemma II.4. Therefore step c in **Reducing** is correct. Step d of **Reducing** is correct by Observation II.8.

Consider the operations in the algorithm **VC**. Step 0 is correct since if $|G|>0$ and $k=0$, G does not have a vertex cover of size bounded by k (assuming G does not consist of isolated vertices). Step 1 is correct by the above discussion of the subroutine **Reducing**. Step 2 simply picks a structure Γ of highest priority in $\mathcal{T}$. By the definition of a good pair, a good pair always exists in the graph as long as the graph is not empty. Hence the algorithm in step 2 will pick a structure Γ. Let us look at step 3 of the algorithm. First observe that each structure in $\mathcal{T}$ is either a 2-tuple, a good pair, or a vertex of degree at least 7. Therefore, one of the condition in step 3 will apply to Γ and the algorithm branches accordingly. In all the cases in step 3 the algorithm branches on z, and hence the branch is valid by the definition of branching on a vertex. What is left is showing that the tuples added in each branch are valid tuples. The tuple created in the first branch is valid by Observation II.6, and the tuple created by the second branch in step 3 is valid by Observation II.7. This completes the proof. $\square$

D. Analysis of the Algorithm

In this section we analyze the running time of the algorithm. The algorithm is a branch-and-bound process and its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of root-leaf paths, or equivalently the number of leaves in the search tree, multiplied by the time spent along each such path. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. Let $F(k)$ be the number of leaves in the search tree of the algorithm when called on the instance (G, k).

First, we derive an upper bound on the number of leaves $F(k)$ of the search tree. This is the main theorem of this chapter whose proof appears in Section E.

Theorem II.11 (The Main Theorem) *For any constant $c \geq 1.2738$, the search tree of the* **VC** *on an instance (G, k) where G is a connected graph, has at most $F(k)$ leaves where $F(k) \leq c^k$.*

Proof. See Theorem II.15, Section E. $\square$

Theorem II.12 *The algorithm* **VC** *solves the VC problem in $O(1.2738^k + kn)$ time.*

Proof. Let (G, k) be an instance of VC. By Theorem II.10 the algorithm **VC** solves the VC problem correctly. Let T be the search tree of the algorithm on the instance (G, k), and let $F(k)$ be the number of leaves in T. If G is connected, then by Theorem II.11, the number of leaves in T is bounded by 1.2738^k. If G is not connected, suppose that G has two connected components G_1 and G_2. (If G has more than two connected components, the statement follows by an inductive argument.) The algorithm can be called recursively on G_1 and G_2. If any of G_1 or G_2, has at most c' vertices for a pre-specified constant c' (picking $c' = 16$ will work), we can compute the size of a minimum vertex cover in constant time by brute-force and without any branching, and the search tree corresponding to this recursive call has one leaf. For example, if $|G_1| \leq 16$ and is not empty, the size of a minimum vertex cover in G_1 is at least 1. Therefore if G has a minimum vertex cover of size at most k, then the size of a minimum vertex cover for G_2 should be at most $k-1$, and the parameter passed in the recursive call to G_2 is $k-1$. We get $F(k) \leq 1+F(k-1) \leq 1+1.2738^{k-1} \leq 1.2738^k$ (note that we can assume that $k \geq 8$ otherwise the algorithm would compute the size of the minimum vertex cover of G by brute-force). On the other hand if $|G_1| > c' = 16$ and $|G_2| \geq 16$, then since **Reducing** does not apply to G, and hence does not apply to G_1 and to G_2, by Proposition II.5, the size of a minimum vertex cover for G_1 is at least $|G_1|/2 \geq c'/2 \geq 8$, and the size of a minimum vertex cover for G_2 is at least $|G_2|/2 \geq 8$. Therefore in the recursive calls of the algorithm to G_1 and G_2 we can pass the parameter $k - 8$. This gives $F(k) \leq 2F(k-8) \leq 1.2738^k$. This shows that the number of leaves in T is bounded by 1.2738^k.

Now let us analyze the time spent along each root-leaf path in T. By Proposition II.1, the number of vertices in G is at most $2k$. Since in each branch the algorithm can create at most one tuple, and since along any root-leaf path of T the algorithm branches at most k times (since each branch decrements k by at least 1), the number of tuples created by the branches of the algorithm is $O(k)$. Now step a.1 and a.2 in **Reducing** can decompose the tuples created by the algorithm thus creating new tuples. Observe that if the algorithm creates a tuple (S, q) in a branch then $q = 2$, and that any decomposition of a tuple decrements q by 1 and when $q = 0$ the tuple is removed. Based on these observations, it can be easily shown that each tuple (S, q) may lead to the creation of at most $O(|S|)$ new tuples, each of them can no longer be decomposed. Since each created tuple has the form (S, q) where $S = N(w)$ for some vertex w, and since $|G| \leq 2k$, we have $|S| \leq 2k$. This means that each tuple can create at most $O(k)$ new tuples, and the total number of tuples along any root-leaf path is $O(k^2)$. Therefore step a of **Reducing** can be implemented to run in $O(k^3)$ time. By Proposition II.5, **General-Fold** runs in $O(k^2\sqrt{k})$ time. All the other operations in **Reducing** and in the algorithm, including the implicit maintenance of the structures in $\mathcal{T}$ and their priorities, can be implemented to run in $O(k^3)$ time using suitable data structures. Therefore the amount of time spent along each node of the search tree is $O(k^3)$, and hence along every root-leaf path of T is $O(k^4)$.

Before any branching node in the search tree **General-Fold** does not apply (because **Reducing** does not apply). By Proposition II.5, the size of the graph before any branching operation is bounded by twice the size of the parameter. Note that this is also true at the root of the tree T by Proposition II.1. By the standard analysis using the *interleaving technique* introduced by Niedermeier and Rossmanith [43], the running time of the algorithm is bounded by $O(1.2738^k + kn)$, where the kn factor is due to the application of Proposition II.1 to the original instance of the problem. This completes the proof. $\square$

Using Theorem V.10, and the fact that the size of a minimum vertex cover in a graph of degree bounded by 6 is bounded by $5n/6 + 1$ (n is the number of vertices in the graph), we get the following theorem.

Theorem II.13 *The Independent Set problem on graphs of degree bounded by 6 can be solved in $O(1.224^n)$ time, where n is the number of vertices in the graph.*

Theorem II.13 improves the previous best polynomial-space algorithm for Independent Set on graphs of degree at most 6 by Robson [28], which runs in time $O(1.227^n)$.

E. Proof of The Main Theorem

In this section, we give a complete proof of Theorem II.11. First, we have the following proposition which will be useful in the proof.

Proposition II.14 *Let v be a vertex that satisfies the statement in step a.4 in* **Reducing**. *If the algorithm does not reject the instance (along this path of the search tree) then v must be included in the cover before any branching operation by the algorithm. Moreover, each recursive call to* **Reducing** *before v is included in the cover, results in the execution of step a.4 of* **Reducing** *that includes a vertex in the cover.*

Proof. By looking at the algorithm **VC** the algorithm only branches when **Reducing** is not applicable. Moreover, since step a.4 in **Reducing** invokes the algorithm recursively, which in turn invokes **Reducing**, steps b–d of **Reducing** will not apply as long as step a.4 is applicable to a vertex in G.

Now suppose that there exists a vertex v and a tuple (S, q) such that $|N(v) \cap S| \geq |S| - q + 1$, and that the algorithm does not reject. When **Reducing** is applied, step a.4 is checked. Since v satisfies this step, if v is considered in this step then v will be included. Now suppose that another vertex $x \neq v$ to which this step applies is checked, and x is included in the cover. If $x \notin S$, then (S, q) is unaffected by the inclusion of x, and v still satisfies this step in the (nested) recursive call to **Reducing** (note that this is true even when $x \in N(v)$). If $x \in S$, then each tuple containing x, and in particular S, will be updated. The tuple (S, q) will be updated to become $(S' = S - \{x\}, q)$. Since $|S| = |S'| + 1$, $|N(v) \cap S'| \geq |N(v) \cap S| - 1 \geq |S| - q \geq |S'| - q + 1$, and step a.4 is still applicable to v in the nested recursive call to **Reducing**. This shows that v will be included in the cover ultimately, and that each preceding call to **Reducing** before v is included, will include one vertex in the cover by step a.4. $\square$

Theorem II.15 *For any constant $c \geq 1.2738$, the search tree of the* **VC** *on an instance (G, k) where G is a connected graph, has at most $F(k)$ leaves where $F(k)$ is upper bounded by the following.*

1. *c^{k-1} if step a.4 or any of steps b–d of* **Reducing** *is applicable.*
2. *$c^{k-1.536}$ if there is a strong 2-tuple structure.*
3. *c^{k-1} if there is a 2-tuple structure.*
4. *c^{k-1} if G is 3-regular.*
5. *$c^{k-0.897}$ if there exist three non-adjacent degree-3 vertices in G such that the three of them do not share a common neighbor.*
6. *c^{k-1} if G has a degree-3 vertex u such that all the vertices in $N(u)$ are of degree 5, and no two vertices in $N(u)$ share a common neighbor other than u.*
7. *$c^{k-0.605}$ if the algorithm picks a good pair (u, z) such that z is almost-dominated by a vertex in $N(u)$.*
8. *$c^{k-0.605}$ if G has a degree-3 vertex u with at least one vertex in $N(u)$ of degree at least 5.*
9. *$c^{k-0.536}$ if G has a degree-3 vertex.*
10. *$c^{k-0.450}$ if G has a degree-4 vertex u such that at least three vertices in $N(u)$ have degree-5, and such that the graph induced by $N(u)$ contains an edge.*

11. *$c^{k-0.450}$ if G has a degree-4 vertex u such that all the vertices in $N(u)$ are of degree 5 and no two of them share a common neighbor other than u.*
12. *$c^{k-0.302}$ if G has a vertex of degree at least 8.*
13. *$c^{k-0.255}$ if G has a degree-4 vertex.*
14. *$c^{k-0.116}$ if G has a degree-5 vertex with at least one degree-6 neighbor.*
15. *c^k in all other cases.*

PROOF. The proof is by induction on the size of the instance (G, k). Assume inductively that *all* the above statements are simultaneously true for any instance (G', k') where $|G'| < |G|$ and $k' < k$.

Before we prove the statements of the theorem we give some general remarks. First, if during the proof we showed that the graph contains a structure Γ with an inductively proven upper bound on the number of leaves when the structure Γ exists in the graph, then even if the algorithm does not pick Γ to process, this upper bound is still valid since the algorithm always picks a structure with the highest priority, and as it will be shown by the statements of the theorem, a structure of higher priority corresponds to a smaller upper bound on the number of leaves in its corresponding search tree. Therefore whenever a certain structure is present in the graph, we can safely claim the upper bound on the number of leaves corresponding to this structure that was inductively proved. Second, if the algorithm branches by reducing the parameter by a value p along one side, and along the other side the algorithm rejects without doing any branching, then the number of leaves in the search tree satisfies $F(k) \leq F(k-p) + 1$. Now we are ready to prove the theorem.

Part 1. Since **Reducing** consists of non-branching operations, and since step a.4 and each of steps b–d include at least one vertex in the cover, we have $F(k) \leq$

$F(k-1) \leq c^{k-1}$, by the inductive hypothesis.

Part 2. Suppose that there is a strong 2-tuple ($S = \{u, z\}$, q=1). Suppose first that **Reducing** applies to G. Note that steps a.2 and a.3 of **Reducing** will not affect this 2-tuple because $q = 1$. If step a.4 of **Reducing** applies, then a vertex v is included in the cover thus reducing the parameter k by 1. Observe that in the resulting instance $S = \{u, z\}$ remains a 2-tuple (not necessarily a strong 2-tuple) since $d(u) \geq 2$ and $d(z) \geq 2$, $q = 1$, and u and z are non-adjacent. If $F(k')$, where $k' = k - 1$, is the number of leaves in the search tree of the resulting instance, then inductively by part (3) of the theorem, $F(k') \leq c^{k'-1} = c^{k-2}$. It follows that $F(k) \leq F(k') \leq c^{k-2} \leq c^{k-1.536}$.

Now if step b in **Reducing** applies, then a vertex v is included in the cover reducing the parameter k by 1. Since both u and z have degree at least 2, v is distinct from u and z. The only way v could affect the strong 2-tuple is when v is a neighbor of u or z, say u. If this is the case then u is included in the cover and now $S = \{z\}$ and $q = 1$. When the algorithm is called recursively in this step the neighbors of z will be included by step a.4 in **Reducing** (since any neighbor of z will satisfy the statement in step a.4). Since $d(z) \geq 2$, and u and z did not share any neighbors (because step a did not apply), at least two vertices will be included in the cover. This is a total reduction in the parameter of value at least 3 giving $F(k) \leq c^{k-3} \leq c^{k-1.536}$. If the removal of v does not affect the strong 2-tuple, the strong 2-tuple will remain in the resulting graph. Letting $F(k')$ be the number of leaves in the resulting search tree, we have $F(k') \leq c^{k'-1.536}$ by induction. Hence $F(k) \leq F(k') \leq c^{k'-1.536} \leq c^{k-2.536}$.

If step d of **Reducing** is applicable, the analysis is similar to the case when step b applies. The only way that the removal of this vertex can affect the strong 2-tuple is when the vertex is one of the two vertices in the tuple, or a neighbor of a vertex in the tuple. The same analysis performed above gives the bound.

Now suppose that step c of **Reducing** applies. If **General-Fold** is applicable, then the subroutine will always reduce the parameter k. If it reduces the parameter k by at least 2, then we have $F(k) \leq F(k-2) \leq c^{k-2} \leq c^{k-1.536}$. If the subroutine reduces the parameter by 1, then the subroutine simply folds a degree-2 vertex w. If w is one of $\{u, z\}$, say u, then since u and z are non-adjacent, z will remain in the resulting graph. Since $d(u) = 2$, by the definition of a strong 2-tuple, $d(z) = 2$ or $d(z) = 3$. By induction, the former case leads to a further reduction in the parameter by value at least 1 by part (1) of the theorem, and the latter case to a reduction of the parameter of value 0.536 by part (9) of the theorem. Therefore the total reduction of the parameter is at least 1.536 and $F(k) \leq c^{k-1.536}$ as required. Now if $w \notin \{u, z\}$, then w cannot be adjacent to both u and z by step a.4 of **Reducing**. Folding w in this case will leave at least one vertex in $\{u, z\}$, and will similarly lead to a total reduction of the parameter of value at least 1.536. If the **Conditional_Struction** operation applies and destroys the strong 2-tuples, then from the way the operation works, the operation must apply to a degree-3 vertex w such that w is in a strong 2-tuple. Note that this operation reduces the parameter by 1. Without loss of generality, assume that the strong 2-tuple containing w is $\{u, z\}$, and suppose that $w = u$. Since u and z are non-adjacent and do not share any neighbors, the operation will not affect the degree of z by Remark II.3, and a similar analysis to the above cases goes through.

Suppose now that **Reducing** is not applicable. In this case we have $d(u) > 2$ and $d(z) > 2$. Since there is a strong 2-tuple, from the way the list of priorities was defined, a strong 2-tuple must be picked by the algorithm as the structure Γ. The algorithm branches in this case on the vertex z. Note that since **Reducing** is not applicable, u and z do not share any neighbors. Suppose first that $d(u) \geq 4$ and $d(z) \geq 4$. Now on the side of the branch where z is included, z is removed from the tuple S and q is kept unchanged. The recursive call to the algorithm will invoke **Reducing** and the neighbors of u will be included in the cover by step a.4 of **Reducing**. Therefore this side of the branch reduces the parameter by at least 5 ($N(u) \cup \{z\}$ are included in the cover). On the other side of the branch $N(z)$ is included reducing the parameter by at least 4. It follows that $F(k) \leq F(k-4) + F(k-5) \leq c^{k-4} + c^{k-5} \leq c^{k-1.536}$.

If $d(u) = d(z) = 3$, then since the **Conditional_Struction** is not applicable, there are no edges between vertices in $N(u)$ and similarly for $N(z)$. Let $N(u) = \{u_1, u_2, u_3\}$ and $N(z) = \{z_1, z_2, z_3\}$. Suppose that there exists a vertex in $N(u)$, say u_1, such that $|N(u_1) - N(z)| \leq 2$. In the side of the branch where the algorithm excludes z and includes $N(z)$, u_1 becomes of degree 1 or 2, and when the subroutine **Reducing** is called the parameter will be further reduced by at least 1. Therefore in this side of the branch the parameter has been reduced by at least $|N(z)| + 1 = 4$. On the other side of the branch where the algorithm includes z, all the vertices in $N(u)$ will be included when **Reducing** is called by step a.4. Moreover, the algorithm creates the tuple $(N(z), 2)$. We first claim that at least two vertices in $N(z)$ do not become isolated in the resulting graph along this side of the branch. Suppose not, then two of the vertices in $N(z)$, say z_1 and z_2 become isolated in $G - (\{z\} \cup N(u))$. Since u and z do not share any neighbors, z_1 and z_2 are only connected to $N(u) \cup \{z\}$. But then $I = \{u, z_1, z_2\}$ is an independent set whose set of neighbors $N(I) = \{z\} \cup N(u)$ satisfies $|N(I)| = |I| + 1$, and **General-Fold** (and hence **Reducing**) is applicable, a contradiction. Therefore two non-isolated vertices in $N(z)$, that are also non-adjacent, will remain in the resulting graph. These two vertices will create a 2-tuple by step a.2 of **Reducing** when applied to the tuple $(N(z), 2)$ created by this side of the branch. This leads to a further reduction of the parameter by at least 1 by part (2) of the theorem. Therefore, along this side of the branch the parameter is reduced by at least 5. It follows that $F(k) \leq F(k-4) + F(k-5) \leq c^{k-4} + c^{k-5} \leq c^{k-1.536}$ as required.

Suppose now that $d(u) = d(z) = 3$ and that the above case does not apply. On the side of the branch where z is excluded, $N(z)$ is included and a degree-3 vertex u remains in the graph. By induction, and by part (9) in the theorem, the number of leaves in the search tree along this side of the branch is bounded by $F(k - 3.536)$. On the other side of the branch, $\{z\} \cup N(u)$ are included in the cover and the tuple $(N(z), 2)$ is created. When **Reducing** is called, it will end up creating a 2-tuple for every two vertices in $N(z)$ (note that no two vertices in $N(z)$ are adjacent because **Conditional_Struction** is not applicable). We claim that at least one of these 2-tuples is a strong 2-tuple. To see this, observe that all the vertices in $N(z)$ have degree at least 2 in the resulting graph. This is true because otherwise a neighbor of z would be almost-dominated by u, and by the way the algorithm branches on 2-tuples, u will be picked by the algorithm instead of z and the previous discussion applies. If $\{z_1, z_2\}$ is not a strong 2-tuple, then one vertex in $\{z_1, z_2\}$, say z_1, has degree at least 4 and the other vertex z_2 has degree at most 3. Now if z_3 has degree at least 4 then $\{z_1, z_3\}$ is a strong 2-tuple, otherwise, $\{z_2, z_3\}$ is a strong 2-tuple. By induction, the number of leaves in the search tree resulting from this side of the branch is at most $F(k - 5.536)$ (since $\{z\} \cup N(u)$ were included and there is a strong 2-tuple). It follows that the number of leaves in the search tree is $F(k) \leq F(k - 3.536) + F(k - 5.536) \leq F(k - 1.536)$.

Part 3. Let $(S = \{u, z\}, q = 1)$ be a 2-tuple. Since $q = 1$ and u and v are non-adjacent, the only way **Reducing** can destroy this 2-tuple is if step a.4, or if one of steps b–d applies. Each of these steps reduces the parameter by at least 1 and $F(k) \leq F(k - 1) \leq c^{k-1}$ as desired.

Now we can assume that **Reducing** is not applicable. This implies that $d(u) \geq 3$ and $d(z) \geq 3$.

If there is a neighbor u' of u such that $|N(u') - N(z)| \leq 2$, then by a similar token to the above, on the side of the branch where $N(z)$ is included u' becomes of degree at most 2, and **Reducing** will further decrease the parameter by at least 1. Therefore along this side of the branch the parameter is decreased by at least $d(z)+1 \geq 4$. On the other side of the branch we include $\{z\} \cup N(u)$ and the parameter is again decreased by at least 4 (note that $d(u) \geq 3$). Therefore $F(k) \leq 2F(k-4) \leq 2c^{k-4} \leq c^{k-1}$.

Suppose now that the above case does not apply and $d(u) = d(z) = 3$. On the side of the branch where the algorithm includes z, $N(u)$ is included and the tuple $(N(z), 2)$ is created. By a similar argument to the above, when **Reducing** is called this tuple will create a 2-tuple (note that every vertex in $N(z)$ has degree at least 2). Inductively, the number of leaves along this side of the branch is bounded by $F(k-5)$ (note that $|\{z\} \cup N(u)| \geq 4$). On the side of the branch where z is excluded we include $N(z)$. It follows that $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$.

In the remaining cases we must have $d(u) > 3$ or $d(z) > 3$. On one side of the branch z and $N(u)$ are included, and on the other side of the branch $N(z)$ is included. This gives us a worst-case bound $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$ as required.

Part 4. Suppose that G is 3-regular. If **Reducing** applies then the parameter is reduced by at least 1 and $F(k) \leq c^{k-1}$ as required. Now suppose that **Reducing** is not applicable. In this case for every degree-3 vertex u no edges exist in the subgraph induced by $N(u)$ (this follows from the inapplicability of **Conditional_Struction**). If there is a 2-tuple (or a strong 2-tuple) then the statement follows from above. The algorithm branches on a good pair (u, z). On the side where z is included the three neighbors of z become of degree 2, and no two of them are adjacent. Then on this side of the branch **Reducing** will apply at least twice reducing the parameter by at least 2. On the side of the branch where $N(z)$ is included, we claim that there must exist at least four non-isolated vertices of degree at most 2. To see why this is the case let $N(z) = \{u, z_1, z_2\}$ and note that $N(z)$ is an independent set. Let B be the set of vertices of degree at most 2 in the graph $G - N(z)$. If $|B| < 4$, then the set $N(z)$ has at most 4 neighbors namely the vertices in $\{z\} \cup B$, and **General-Fold** would be applicable, a contradiction. Therefore, $|B| \geq 4$. Now if a vertex $w \in B$ is isolated in $G - N(z)$ then the set of vertices $I = \{z, w\}$ has the neighboring set $N(I) = N(z)$ with $|N(I)| = |I| + 1$, and again **General-Fold** would be applicable. Therefore the graph $G - N(z)$ contains at least four non-isolated vertices of degree at most two. Again, in this side of the branch **Reducing** will be applied twice totally reducing the parameter along this side of the branch by at least 5. It follows that $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$ as required.

Part 5. Let x_1, x_2, x_3 be three degree-3 vertices in G such that no two of them are adjacent and such that the three vertices do not share a common neighbor. If the graph is 3-regular then the statement follows from part (4) above. If **Reducing** is applicable, or if there exists a 2-tuple, then the statement follows either from the fact that **Reducing** reduces the parameter by at least 1, or from the parts (2) and (3) of the theorem. If this is not the case, then from the priority list of the structures, the structure Γ picked by the algorithm must be a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 3$ (note that the minimum degree of a vertex in the graph is 3). Suppose first that z is almost-dominated by a vertex $v \in N(u)$.

If $d(z) = 3$ let $N(z) = \{u, z_1, z_2\}$ be the neighbors of z and observe that since **Conditional_Struction** does not apply, no two vertices in $N(z)$ are adjacent. The algorithm will branch on z. If in the side of the branch where z is included the algorithm rejects before doing any branching, then we have $F(k) \leq F(k-3) + 1 \leq c^{k-0.897}$ as desired. Suppose now that this is not the case. On the side of the branch where z is included the algorithm will create the tuple $(\{u, z_1, z_2\}, 2)$ which will immediately be decomposed by step a.2 of **Reducing** into the tuples $(\{u, z_1\}, 1)$,

$(\{u, z_2\}, 1)$, $(\{z_1, z_2\}, 1)$. Since z is almost-dominated by v, v is adjacent to all vertices in $N(z)$ except at most 1. Therefore there exists a tuple $(S, 1)$ among $(\{u, z_1\}, 1)$, $(\{u, z_1\}, 1)$, $(\{u, z_1\}, 1)$ such that step a.4 of **Reducing** applies to v and $(S, 1)$, and v will be included in the cover. By Proposition II.14, all preceding recursive calls to **Reducing** end up executing step a.4 of **Reducing** and include vertices in the cover. If these recursive calls include two vertices in the cover before v is included, then this side of the branch ends up including at least four vertices in the cover (including z and v) and hence reducing the parameter by at least 4. Suppose that exactly one vertex y is included in these recursive calls before v is included. If $y = u$, then the tuple $(\{u, z_1\}, 1)$ will be updated to $(\{z_1\}, 1)$ and step a.4 of **Reducing** will be applicable to all vertices in $N(z_1)$. Since $u \notin N(z_1)$, this means that at least four vertices will be included in the cover, namely $N(z_1) \cup \{u\}$, in this side of the branch. Now if $y \neq w$, where w is the third neighbor of u, then after v is included, u will be a degree-1 vertex and **Reducing** will end up further reducing the parameter by at least one, again yielding a total reduction of the parameter by 4. Suppose now that w is the vertex that is included before v is included. Now in the resulting graph after v is included, $(\{z_1, z_2\}, 1)$ is a 2-tuple. To see why this is the case, note that z_1 and z_2 are non-adjacent and none of them can become isolated in $G - \{u, v, w, z\}$, otherwise, **General-Fold** would be applicable to the set consisting of u and that vertex. By part (3) of the theorem, this reduces the parameter by 1 yielding a total reduction of the parameter by 4 along this side of the branch. On the other side of the branch $N(z)$ is included. Notice that along this side of the branch a non-isolated vertex of degree at most three must remain in the graph because there were three non-adjacent degree-3 vertices in the graph that do not share a common neighbor. One of these vertices must remain and cannot be isolated (otherwise **General-Fold** will apply to this vertex and z). If this vertex has degree one or two, then **Reducing** will end up reducing the parameter by at least one. If this vertex has degree three, then by part (9) of the theorem, a further reduction of the parameter by value 0.536 can be claimed. Therefore along this side of the branch we can claim a reduction of the parameter of value at least 3.536. It follows that $F(k) \leq F(k-3.536) + F(k-4) \leq c^{k-3.536} + c^{k-4} \leq c^{k-0.897}$ as claimed.

Suppose now that $d(z) \geq 4$. By a similar argument to the above, we can claim that along the side of the branch where z is included v satisfies step a.4 of **Reducing**. A similar (and easier) argument using Proposition II.14 will show that either **Reducing** ends up including a total of three vertices along this side and leaving a vertex of degree three in the resulting graph, allowing us to claim a further reduction of value 0.536 in the parameter by part (9) of the theorem, or it will include at least four vertices. Therefore a total reduction in the parameter of value at least 3.536 can be claimed along this side of the branch. On the other side of the branch $N(z)$ is included reducing the parameter by at least 4 and the result follows using the same argument as above.

Suppose now that z is not almost-dominated by any vertex in $N(u)$. Then from the choice of a good pair and the fact that G is not regular, we have $d(z) \geq 4$. The algorithm now branches on z and in the side where $N(z)$ is included the algorithm will create a tuple $(N(u), 2)$. Suppose first that $d(z) = 4$. On the side of the branch where z is included, u becomes a degree-2 vertex and **General-Fold** is applicable to u. Any operation in **Reducing**, other than **General-Fold** will leave u in the resulting graph a non-isolated vertex of degree at most 2, and **Reducing** will still be applicable (note that if **General-Fold** is applicable but was not applied then **Conditional_Struction** was not applied as well by the respective order of these two operations in the algorithm). This will reduce the parameter by at least 3 along this side of the branch. If instead **General-Fold** was applied, then it is easy to see that since no two of x_1, x_2, and x_3 are adjacent, and since they do not share a common neighbor, at least one of them will remain a non-isolated vertex of degree at most 3 in the graph resulting from including z and applying **General-Fold** to a degree-2 vertex (if **General-Fold** was applied to a set I with $|I| > 1$, then we can claim a reduction in the parameter of at least 2 from this operation). This is true since z cannot be one of the vertices in $\{x_1, x_2, x_3\}$ since $d(z) = 4$. This will lead to a further reduction in the parameter of value at at least 0.536 by part (9) of the theorem giving a total reduction along this side of the branch of value at least 2.536. On the other side of the branch where $N(z)$ is included the tuple $(N(u), 2)$ will result by step a.2 of **Reducing** in the strong 2-tuple $(\{v, w\} = N(u) - \{z\}, 1)$. To see why $\{v, w\}$ is a strong 2-tuple observe that $\{v, w\}$ are non-adjacent since $d(u) = 3$ and **Conditional_Struction** does not apply. Moreover, from the choice of the good pair (u, z) and since z is not almost-dominated by any vertex in $N(u)$, none of the vertices v or w is can be almost-dominated by z (otherwise that vertex would be chosen in place of z). It follows that the degree of v and w in the resulting graph is at least two. Moreover, the degree of these two vertices in G was bounded by 4 since $d(z) = 4$ and by the choice of z, z had the maximum degree among the neighbors of u. It follows that the degrees of v and w in $G - N(z)$ is bounded by 3 (since u was removed). This shows that $\{v, w\}$ is a strong 2-tuple. By part (2) of the theorem this gives a further reduction of the parameter of value at least 1.536, giving a total reduction of value at least 5.536 along this side of the branch. It follows that $F(k) \leq F(k - 2.536) + F(k - 5.536) \leq c^{k-2.536} + c^{k-5.536} \leq c^{k-0.897}$ as required.

Suppose now that $d(z) \geq 5$. By a similar argument to the above, on the side of the branch where z is included, u becomes a degree-2 vertex and **General-Fold** is applicable. Moreover, if **Reducing** does not apply **General-Fold** then u will remain and **Reducing** will be applicable again claiming a total reduction in the parameter of value at least 3. If **Reducing** applies **General-Fold** then a non-isolated vertex of degree at most three remains claiming a total reduction in the parameter of value at least 2.536 along this side of the branch. On the other side of the branch where $N(z)$ is included $\{v, w\}$ become a 2-tuple (not necessarily a strong 2-tuple) yielding a further reduction in the parameter of value at least 1 by part (3) of the theorem. It follows that $F(k) \leq F(k-2.536) + F(k-6) \leq c^{k-2.536} + c^{k-6} \leq c^{k-0.897}$ as required.

Part 6. Let Γ be a structure of highest priority picked by the algorithm. If Γ is a 2-tuple (or a strong 2-tuple) the the statement follows from the previous parts of the theorem. If this is not the case, then by the priority list of the algorithm, Γ is a good pair (u, z) such that $d(u) = 3$, and all the neighbors of u, say $\{v, w, z\}$ are degree-5 vertices such that no two of them share a common neighbor other than u. If two vertices in $\{v, w, z\}$ are adjacent, then **Conditional_Struction** is applicable, and hence **Reducing** is applicable which reduces the parameter by at least 1 implying the desired result. Suppose that this is not the case. Since z is not almost-dominated by any vertex in $N(u)$ (no two vertices in $N(u)$ share a common neighbor other than u), the algorithm will branch on z by including z on one side of the branch, and excluding it and creating a tuple $(N(u), 2)$ on the other side of the branch. On the side of the branch where z is included, u becomes of degree two. If **Reducing** does not apply **General-Fold** to u, then u will remain in the graph (again note that **Reducing** did not apply **Conditional_Struction** by the way the algorithm works) non-isolated and having degree at most two. this means that **Reducing** will also be applicable and a total reduction of at least 3 in the value of the parameter can be claimed along this side of the branch. If **Reducing** applies **General-Fold** to a vertex other than u, then again a reduction of value 2 can be claimed if **General-Fold** applies to a set I of cardinality at least two, or if **General-Fold** applies to another degree-2 vertex since u will remain (none of the neighbors of u could be the vertex folded since each has a degree larger than two). If **General-Fold** applies to u, then a vertex of degree eight results from folding u, and by part (12) of the theorem, an additional reduction of the parameter of value at least 0.302 can be claimed. Therefore along this side of the branch we can claim a reduction of the parameter of value at least 2.302. On the side of the branch where $N(z)$ is included, the tuple $(N(u), 2)$ will be decomposed into the tuple $(\{v, w\}, 1)$ in step a.2 of reducing. Since v and w are non-adjacent and have degree exactly 4 in the resulting graph (since no two neighbors of u share a common neighbor besides u), this tuple is a strong 2-tuple giving a further reduction in the parameter of value at least 1.536. Therefore the total reduction along this side of the branch is at least 6.536 and $F(k) \leq F(k-2.302)+F(k-6.536) \leq c^{k-2.302}+c^{k-6.536} \leq c^{k-1}$ as required.

Part 7. Suppose the algorithm picks a structure Γ such that Γ is a good pair (u, z) and z is almost-dominated by a vertex $v \in N(u)$. Note that **Reduce** is not applicable at this point.

Suppose that $d(u) = 3$. Then all vertices in $N(u)$ have degree at least 3, and no two of them are adjacent (since **Conditional_Struction** is not applicable). If $d(z) = 3$ let z_1 and z_2 be the other neighbors of z. On the side of the branch where z is included, the algorithm forms the tuple $(N(z), 2)$ which will be decomposed into the tuples $(\{u, z_1\}, 1)$, $(\{z_1, z_2\}, 1)$, $(\{u, z_2\}, 1)$, by step a.2 of the algorithm. Now since z is almost-dominated by v, v and at least one tuple S among these three tuples will satisfy step a.4 in **Reducing**. By Proposition II.14, v will be included before any branching by the algorithm. If **Reducing** includes two vertices before v, then the total reduction in the parameter along this side of the branch is at least 4. If **Reducing** includes exactly one vertex before v, let this vertex be y. If $y \in \{u, z_1, z_2\} = N(z)$, then the neighbors of the vertices in $N(z) - \{y\}$ will be included by step a.4 of **Reducing** when applied the the vertices in $N(z) - \{y\}$ and the three tuples formed above. Note that the set $N(z) - \{y\}$ has at least two neighbors in the graph $G - \{z, y\}$ (otherwise **General-Fold** applies). Therefore the parameter is reduced by at least 4 in this case. If $y \notin \{u, z_1, z_2\}$, then $(\{z_1, z_2\}, 1)$ remains a 2-tuple in the resulting graph when y and v are included and a reduction in the parameter of value at least 1 can be claimed by part (3) of the theorem (note that none of z_1, z_2 can be isolated in the resulting graph because **General-Fold** does not apply). Suppose now that **Reducing** includes v in the next execution. Let w be the other vertex in $N(u)$. When v is included, u becomes a degree-1 vertex, and **Reducing** is still applicable. If **Reducing** in the following execution does not include u or w, then u remains a degree-1 vertex in the resulting graph, and **Reducing** will still be applicable. On the other hand, if **Reducing** includes u or w in the next execution, then $\{z_1, z_2\}$ remains a 2-tuple in the resulting graph, and a further reduction in the parameter of value 1 can be claimed. It follows that in all cases this side of the branch will reduce the parameter by at least 4. On the other side of the branch the algorithm includes $N(z)$ reducing the parameter by 3. It follows that $F(k) \leq F(k-4) + F(k-3) \leq c^{k-3} + c^{k-4} \leq c^{k-0.605}$ as required.

Suppose now that $d(u) = 3$ and $d(z) \geq 4$. By a similar argument to the above we can show that on the side of the branch where z is included step a.4 applies to v and we can show that along this side of the branch the total reduction in the parameter is at least 3. On the other side of the branch $N(z)$ is included yielding a reduction in the parameter of value at least 4, and the statement follows as in the above case.

Suppose now that $d(u) = 4$. In this case $d(z) \geq 4$. Similar to the above analysis, when z is included step a.4 applies to v. If another vertex is included before v we get a reduction in the parameter of value 3; otherwise v is included and u become of degree 2 yielding a further reduction in the parameter of value at least 1 by **Reducing**. Therefore we get a total reduction in the parameter of value at least 3 along this side of the branch. Along the other side we include $N(z)$ and the parameter is reduced by at least 4. The statement follows.

If $d(u) = 5$ we have $d(z) \geq 5$. When z is included, if v is not included immediately by **Reducing** then the parameter will be reduced by at least 3 along this side. If v is immediately included then u becomes a degree-3 vertex and by part (9) of the theorem, a further reduction of the parameter of value 0.536 can be claimed. Therefore the total reduction in the parameter along this side is at least 2.536. When $N(z)$ is included the parameter is reduced by at least 5. We have $F(k) \leq F(k - 2.536) + F(k - 5) \leq c^{k-2.536} + c^{k-5} \leq c^{k-0.605}$ as required.

If $d(u) \geq 6$, and hence $d(z) \geq 6$, by a similar token to the above, when z is included v will be included reducing the parameter by at least 2. On the other side $N(z)$ is included reducing the parameter by at least 6. Therefore $F(k) \leq F(k-2) + F(k-6) \leq c^{k-2} + c^{k-6} \leq c^{k-0.605}$.

<u>Part 8.</u> Let Γ be the structure with highest priority picked by the algorithm. If Γ is a 2-tuple or a good pair (u, z) such that z is almost-dominated by a neighbor of u, then the statement follows from the above parts of the theorem. If this is not the case, then from the priority list of the structures, the algorithm will pick a good pair (u, z) such that $d(u) = 3$ and $d(z) \geq 5$. Note that since **Reduce** is not applicable no two neighbors of u are adjacent. Let $N(u) = \{v, w, z\}$. Note also that by the choice of z in a nice pair, if z almost-dominates a vertex in $\{v, w\}$ then z must also be almost-dominated by a vertex in $\{v, w\}$, and by part (7) of the theorem the statement follows. Therefore, we can assume that no vertex in $\{v, w\}$ is almost-dominated by z. The algorithm branches on z. When z is included u becomes of degree 2, and **Reducing** is applicable reducing the parameter by at least 1. Therefore the total reduction along this side of the branch is at least 2. On the other side of the branch when $N(z)$ is included the algorithm creates a tuple $(N(u), 2)$ which reduces to $(\{v, w\}, 1)$ by step

a.4 of **Reducing**. Since no vertex in $\{v, w\}$ is almost-dominated by z, v and w have degree at least 2 in the resulting graph and are non-adjacent, and a further reduction of the parameter by value 1 can be claimed by part (3) of the theorem. Therefore $F(k) \leq F(k-2) + F(k-6) \leq c^{k-2} + c^{k-6} \leq c^{k-0.605}$.

Part 9. Suppose that G has a vertex of degree 3. Let Γ be the structure picked by the algorithm. Again, from the previous parts of the theorem, and from the priority list of the structures, we can assume that Γ is a good pair (u, z) where $N(u) = \{v, w, z\}$ such that $d(u) = 3$, $d(v) \leq d(w) \leq d(z) = 4$ (note that by part (4) of the theorem the graph is not 3-regular and is connected by the assumption of the theorem), no vertex among $N(u) = \{v, w, z\}$ is almost-dominated by another by part (7), and no two vertices in $N(u)$ are adjacent since **Conditional_Struction** is not applicable. The algorithm branches on z. On the side where z is included, u becomes of degree 2, and **Reducing** is applicable. Therefore a reduction in the parameter of value at least 2 can be claimed along this side of the branch. On the side of the branch where $N(z)$ is included, the tuple $(N(u), 2)$ is created and will be decomposed into the tuple $(\{v, w\}, 1)$ in step a.2 of **Reducing**. It is easy to see from the above conditions that this is a strong 2-tuple giving a further reduction in the parameter of value at least 1.536. Therefore $F(k) \leq F(k-2) + F(k-5.536) \leq c^{k-2} + c^{k-5.536} \leq c^{k-0.536}$.

Part 10. Suppose that G has a degree-4 vertex such that at least three of its neighbors are of degree 5 and such that the graph induced by this set of neighbors contains an edge. Note that **Reducing** does not apply and hence **Conditional_Struction** does not apply as well. Let Γ be the structure of highest priority picked by the algorithm. By the previous parts of the theorem, and from the priority list of the structures, we can assume that Γ is a good pair (u, z) such that $d(u) = 4$ and at least three vertices in $N(u) = \{v, w, r, z\}$ are of degree 5 and there is an edge among the vertices in $N(u)$. We can also assume by part (7) above and the choice of z in a good pair that no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$. By the choice of z in a good pair, and since at least two vertices in $\{v, w, r\}$ are of degree-5, there must exist an edge among the vertices $\{v, w, r\}$. The algorithm branches on z. In the side where z is included, u becomes a degree-3 vertex with at least one edge among its neighbors, and **Reducing** is applicable (since **Conditional_Struction** is applicable). Therefore we can claim a reduction in the parameter of value 2 along this side of the branch. On the side where $N(z)$ is excluded, since **Conditional_Struction** does not apply to u, and no vertex in $N(u)$ is almost-dominated by another, a non-isolated vertex of degree at most 4 remains in the graph. If this vertex has degree at most 2 then we can claim a reduction of the parameter of value at least 1 by **Reducing**. If this vertex has degree 3 then a reduction in the parameter of value 0.536 can be claimed by part (9) above. If this vertex has a degree 4 vertex then we can claim a reduction in the parameter of value at least 0.255 by part (13). Therefore, along this side of the branch the parameter is reduced by at least 5.255. It follows that $F(k) \leq F(k-2) + F(k-5.255) \leq c^{k-2} + c^{k-5.255} \leq c^{k-0.450}$.

Part 11. Suppose that G has a vertex of degree 4 such that all its neighbors are of degree 5 and no two of them share a common neighbor other than the vertex itself. Let Γ be the structure of highest priority picked by the algorithm. By the above parts of the theorem and by the list of priorities, we can assume that Γ is a good pair (u, z) where $d(u) = 4$, all vertices in $N(u)$ have degree 5, no two vertices in $N(u)$ share a neighbor other than u, no edge exists among the vertices in $N(u)$, and no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$. The algorithm branches on z. In the side of the branch where z is included u becomes a degree-3 vertex with three degree-5 neighbors such that no two of them share a common neighbor other than u. Therefore, by part (6) of the theorem, we can claim a further reduction in the parameter of value at least 1 on this side of the branch. On the side of the branch where $N(z)$ is included a degree-4 vertex remain and we can claim a further reduction in the parameter of value 0.255 by part (13). It follows that $F(k) \le F(k-2) + F(k-5.255) \le c^{k-2} + c^{k-5.255} \le c^{k-0.450}$.

Part 12. Suppose that G has a vertex of degree at least 8. Let Γ be the structure of highest priority picked by the algorithm. If Γ is not a vertex of degree at least 8, then it must have a higher ranking in the list and the above parts of the theorem show that processing such a structure will give a search tree of size $F(k) \le F(k-0.302)$. If Γ is a vertex z with $d(z) \ge 8$, then the algorithm branches on z. We get $F(k) \le F(k-1) + F(k-8) \le c^{k-1} + c^{k-8} \le c^{k-0.302}$.

Part 13. Suppose that G has a degree-4 vertex. Again we can assume that none of the above cases applies. We can assume that the algorithm will pick a good pair (u, z) with $d(u) = 4$ and $d(z) \ge 4$. Let $N(u) = \{v, w, t, z\}$. We can assume that no three edges exist among the vertices in $N(u)$ (otherwise **Conditional_Struction** applies) and no vertex in $N(u)$ is almost-dominated by another. The algorithm branches on z.

Suppose first that G is 4-regular, and note that by the choice of a good pair, we can assume that no vertex is almost-dominated by another, since otherwise a vertex good pair (u, z) will be picked where z is almost-dominated by a vertex in $N(u)$ (because all tag vectors have the same value) and to which part (8) of the theorem applies. Let $N(u) = \{v, w, t, z\}$ and $N(z) = \{z_1, z_2, z_3, u\}$.

Suppose that there is at least one edge among the vertices $\{v, w, t\}$. On the side of the branch where the algorithm includes z, **Reducing** becomes applicable (because **Conditional_Struction** is applicable to u). If **Reducing** does not apply **Conditional_Struction**, then **Reducing** reduces the parameter by at least 1, and a non-isolated vertex of degree at most 3 remains in the graph (namely, a vertex in $\{u, z_1, z_2, z_3\}$), and we can claim a further reduction in the parameter of value at least 0.536 by part (9) of the theorem (or better). Therefore, on the side of the branch a total reduction in the parameter of value at least 2.536 can be claimed. If **Reducing** applies **Conditional_Struction** we will show that a vertex of degree at most 3 remains in the graph and hence a total reduction of value 2.536 can be claimed. Note first that when z is included the only vertices of degree 3 in the graph are u, z_1, z_2, and z_3. Suppose that the struction applies to a vertex y, then y must be a vertex in $N(z)$. Let y_1, y_2, and y_3, be the neighbors of y and assume, without loss of generality, that there is an edge between y_1 and y_2. If two vertices among $\{y_1, y_2, y_3\}$ are of degree 3, then these vertices have to be adjacent to z in G, and there are at least three edges between the vertices in $N(y)$ (note that z is in $N(y)$), which would make **Conditional_Struction** applicable to y in G, and this is not case by our assumption. Therefore, at most one vertex in $\{y_1, y_2, y_3\}$ is a degree-3 vertex. When **Conditional_Struction** is applied to y, at most two degree-3 vertices will be removed. Now if no vertex of degree at most 3 remains in the graph, then by Remark II.3, the operation will only increase the degree of the neighbors of y_3. Therefore at least two neighbors of y_3 other than y (which was removed) must be also neighbors of z, and y_3 and z share at least there neighbors. This means that y_3 is almost-dominated by z, contradicting our assumption. It follows that on this side of the branch a degree-3 vertex remains, and we can claim a reduction in the parameter of value at least 2.536. On the other side of the branch where $N(z)$ is included a non-isolated vertex of degree at most 3 remains in the graph, and a further reduction in the parameter of value at least 0.536 can be claimed by part (9). We get $F(k) \leq F(k - 2.536) + F(k - 4.536) \leq c^{k-2.536} + c^{k-4.536} \leq c^{k-0.255}$.

By the selection of of the vertices u and z in a good pair, we can now assume that for any vertex y, no edge exists between the neighbors of y. Moreover, note that since no vertex is almost-dominated by another vertex, no three vertices can share more than one common neighbor. On the side of the branch where z is included, the vertices z_1, z_2 and z_3 become degree-3 vertices such that no two of them are adjacent, and such that they do not share any common neighbor in $G - z$ (since z is a common neighbor of these vertices). Therefore, by part (9) of the theorem we can claim a further reduction in the parameter of value at least 0.897 totally reducing the parameter by at least 1.897. On the side of the branch where $N(z)$ is included, if one of the vertices in $\{v, w, t\}$ become of degree at most 2 (note that this vertex cannot become isolated since this vertex would be almost-dominated by z) **Reducing** will apply. If all these vertices become of degree 3 in $G - N(z)$, then by a similar token to the above, no two of these vertices are adjacent and they do not share a common neighbor in $G - N(z)$, therefore part (5) applies further reducing the parameter by a value of at least 0.897. We get $F(k) \leq F(k - 1.897) + F(k - 4.897) \leq c^{k-1.897} + c^{k-4.897} \leq c^{k-0.255}$.

Now we can assume that G is not 4-regular. Since G is not connected, we can assume that $d(z) \geq 5$ and $d(v) \leq d(w) \leq d(t) \leq d(z)$ by the choice of z

Suppose that $d(z) \geq 6$. On the side of the branch where z is included u becomes a degree-3 vertex and we can claim a further reduction in the parameter of value at least 0.536. When $N(z)$ is included the parameter is reduced by at least 6. We get $F(k) \leq F(k - 1.536) + F(k - 6) \leq c^{k-1.536} + c^{k-6} \leq c^{k-0.255}$.

Suppose now that $d(z) = 5$. If there is an edge among the vertices in $\{v, w, t\}$, then on the side of the branch where z is included **Reducing** is applicable, and we can claim a further reduction in the parameter of value at least 1. On the other side of the branch $N(z)$ is included. We get $F(k) \leq F(k-2) + F(k-5) \leq c^{k-2} + c^{k-5} \leq c^{k-0.255}$.

If there are exactly two edges between z and two vertices in $\{v, w, t\}$, say w and t, then on the side of the branch where $N(z)$ is included the algorithm creates a tuple $(N(u), 2)$. This tuple will be reduced subsequently to the tuple $(\{v\}, 1)$ since z is excluded from $N(u)$ and t and r are included. By step a.4 of **Reducing** and Proposition II.14, all neighbors of v will be included in the cover. Since v is not almost-dominated by z, the parameter will be decreased further by at least 1. When z is included u becomes of degree 3, and we can claim a further reduction in the parameter of value at least 0.536. We get $F(k) \leq F(k-1.536)+F(k-6) \leq c^{k-1.536}+c^{k-6} \leq c^{k-0.255}$. The analysis is very similar if there is exactly one edge between z and a vertex in $\{v,w,t\}$, say t, because on the side of the branch where z is excluded a 2-tuple will be created namely $\{v,w\}$.

If there exists a vertex in $\{v,w,t\}$ of degree 5, say t, and another vertex of degree 4, say v, then on the side of the branch where z is included, u becomes a degree-3 vertex with a at least one neighbor of degree 5, and we can claim a further reduction in the parameter of value at least 0.605 by part (8). When $N(z)$ is included v becomes a non-isolated vertex of degree at most 3 and we can claim a further reduction in the parameter of value at least 0.536 by part (9). We get $F(k) \leq F(k-1.605)+F(k-5.536) \leq c^{k-1.605}+c^{k-5.536} \leq c^{k-0.255}$.

If all vertices in $\{v,w,t\}$ have degree 4, and if v shares a neighbor other than u with at least one vertex in $\{v,w,t\}$, say t, then on the side of the branch where $N(z)$ is included, t becomes a non-isolated vertex of degree at most 2, and we can claim a further reduction in the parameter of value 1 since **Reducing** will be applicable. On the side where z is included, u becomes of degree 3 and we can claim a further reduction in the parameter of value at least 0.536. We get $F(k) \leq F(k-1.536)+F(k-6) \leq c^{k-1.536}+c^{k-6} \leq c^{k-0.255}$. Suppose now that z does not share any neighbors with $\{v,w,t\}$. If $\{v,w,t\}$ share a common neighbor $y \neq u$, then on the side of the branch where $N(z)$ is included the algorithm will create the tuple $(N(u),2)$, which will be reduced to $(\{v,w,t\},1)$. Now y satisfies step a.4 in **Reducing** with respect to this tuple and hence will be included by Proposition II.14 further reducing the parameter by 1. When z is included u becomes of degree 3. We get $F(k) \leq$

$F(k-1.536)+F(k-6) \leq c^{k-1.536}+c^{k-6} \leq c^{k-0.255}$. Now if v, w, and t do not share any common neighbor, then on the side of the branch where $N(z)$ is included these vertices become three vertices of degree 3 such that no two of them are adjacent and such that the three of them do not share a common neighbor. By part (5), we can claim a further reduction in the parameter of value at least 0.897. When z is included u becomes of degree 3. We get $F(k) \leq F(k-1.536) + F(k-5.897) \leq c^{k-1.536} + c^{k-5.897} \leq c^{k-0.255}$.

Now suppose that all the vertices in $\{v, w, t\}$ are of degree 5. If z shares a neighbor other than u with any vertex in $\{v, w, t\}$, say t, then on the side of the branch where $N(z)$ is included t becomes a non-isolated vertex of degree at most 3 and we can claim a further reduction of the parameter of value at least 0.536 by part (9). When z is included u becomes a degree-3 vertex with at least one degree-5 neighbor and we can claim a reduction in the parameter of value at least 0.605. We get $F(k) \leq F(k-1.605) + F(k-5.536) \leq c^{k-1.605} + c^{k-5.536} \leq c^{k-0.255}$. If z does not share any neighbors with $\{v, w, t\}$ other than u, then by the choice of z in a good pair (since all vertices in $N(u)$ have the same degree), no two vertices in $N(u)$ share a neighbor other than u. This case is actually part (11) in the theorem and we have $F(k) \leq c^{k-0.450} \leq c^{k-0.255}$ as required.

Part 14. Suppose that G has a degree-5 vertex with a neighbor of degree 6. Again if the structure Γ picked by the algorithm is not a good pair (u, z) with $d(u) = 5$ and $d(z) = 6$ then the statement follows from the above parts of the theorem. Suppose now that this is the case. The algorithm branches on z. When z is included u becomes of degree 4 and we can claim a further reduction in the parameter of value at least 0.255 by part (13). When $N(z)$ is included the parameter is reduced by at least 6. We get $F(k) \leq F(k-1.255) + F(k-6) \leq c^{k-1.255} + c^{k-6} \leq c^{k-0.116}$.

Part 15. We can assume in this case that none of the previous parts applies. In particular, part (11) does not apply and the graph has degree bounded by 7. If there exists a vertex z of degree 7, then by looking at the list of priorities, the algorithm will branch on z (or any other vertex of degree 7). This gives $F(k) \leq F(k-1)+F(k-7) \leq c^{k-1}+c^{k-7} \leq c^k$. Suppose now that the graph has degree bounded by 6. By part (1), there are no vertices in the graph of degree 1 and 2. By parts (9) and (13), there are no vertices in the graph of degree less than 5. By part (14), and the fact that G is connected, G is either 5-regular or 6-regular.

Suppose first that G is 6-regular. Since none of the above parts of the theorem applies, the algorithm in this case will pick a good pair (u, z) and branch on z. When z is included u becomes a degree-5 vertex with a neighbor of degree 6, and we can claim a further reduction in the parameter of value at least 0.116 by part (14) of the theorem. When $N(z)$ is included the parameter is reduced by at least 6. We get $F(k) \leq F(k-1.116)+F(k-6) \leq c^{k-1.116}+c^{k-6} \leq c^k$.

Suppose now that G is 5-regular. Again, since none of the above parts applies, the algorithm will pick a good pair (u, z) and branch on z. Let $N(u)=\{v, w, r, t, z\}$. Note that in particular, no vertex in $N(u)$ is almost-dominated by another by part (7).

If z is adjacent to at least two vertices in $\{v, w, r, t\}$, then by the choice of z in a good pair, the graph induced by $\{v, w, r, t\}$ must contain at least three edges. Therefore on the side of the branch where z is included **Reducing** applies (because **Conditional_Struction** applies). On the side of the branch where $N(z)$ is included a vertex of degree at most 4 remains, and a further reduction in the parameter of value at least 0.255 can be claimed by part (13) (or better if the degree is less than 4). We get $F(k) \leq F(k-2)+F(k-5.255) \leq c^{k-2}+c^{k-5.255} \leq c^k$.

If z is adjacent to one vertex in $\{v, w, r, t\}$, then there is at least one edge in the subgraph induced by $\{v, w, r, t\}$. On the side of the branch where z is included, u becomes of degree 4 and at least three of its neighbors are of degree 5 with at least one edge among them. Therefore we can claim a further reduction in the parameter of value at least 0.450 by part (10). On the side of the branch where $N(z)$ is included a vertex of degree at most 4 remains, and a further reduction in the parameter of value at least 0.255 can be claimed by part (13). We get $F(k) \leq F(k-1.450)+F(k-5.255) \leq c^{k-1.450} + c^{k-5.255} \leq c^k$.

If z shares one or more neighbors with a vertex in $\{v, w, r, t\}$, say with t, then when z is excluded t becomes a non-isolated vertex of degree at most 3, and a further reduction in the parameter of value 0.536 can be claimed by part (9). When z is included u becomes of degree 4, and we can claim a further reduction in the parameter of value 0.255 by part (13). We get $F(k) \leq F(k-1.255)+F(k-5.536) \leq c^{k-1.255} + c^{k-5.536} \leq c^k$.

Now from the choice of z in a good pair, we can assume that no two vertices in $\{v, w, r, t, z\}$ share a neighbor other than u. When z is included part (11) applies to u and we can claim a further reduction in the parameter of value at least 0.450. When $N(z)$ is included we can claim a further reduction in the parameter of value 0.255 by part (13). We get $F(k) \leq F(k-1.450) + F(k-5.255) \leq c^{k-1.450} + c^{k-5.255} \leq c^k$.

This completes the proof. $\square$

CHAPTER III

LABELED SEARCH TREE AND AMORTIZED ANALYSIS[*]

In this chapter we extend our study on Vertex Cover. This time, we focus our attention on the techniques of analyzing branch-and-bound algorithms. We propose a different approach to analyzing the size of the search tree. Recall that in the previous chapter, although we utilized a set of powerful techniques including *tuples*, *struction*, and *general folding*, the way we analyzed the search tree is still using the "local" amortized analysis. The goal was to balance each expensive branching operation by combining it with a few more efficient operations that may follow. The ultimate goal, of course, is to balance all branchings in the entire search tree, perhaps using a "global" amortized analysis. In this chapter we take the first step toward that goal by presenting an "almost-global" amortized analysis that balances all branchings in any root-leaf path in a search tree.

In order to illustrate the effectiveness of this new technique, we present a simple algorithm of running time $O(1.194^k + n)$ for the parameterized Vertex Cover problem on degree-3 graphs, and a simple algorithm of running time $O(1.1255^n)$ for the Maximum Independent Set problem on degree-3 graphs. Both algorithms improve the previous best algorithms for the problems. This demonstrates how simple algorithms, if analyzed properly, may perform much better than the upper bounds on their running time derived by considering only a worst-case scenario.

[*]Part of this chapter is reprinted with kind permission of Springer Science and Business Media from Algorithmica, Online First Issue, 2005, pp. OF1–OF29, "Labeled Search Trees and Amortized Analysis: Improved Upper Bounds for NP-Hard Problems", Jianer Chen, Iyad A. Kanj, and Ge Xia. Copyright 2005 by Springer-Verlag.

A. Amortized Analysis on Labeled Search Trees

The most popular technique for solving NP-hard problems precisely is the *branch-and-search* process, which can be depicted by a search tree model described as follows. Each node of the search tree is associated with an instance of the problem. At a node α in the tree the search process considers a local structure in the problem instance associated with α, and enumerates some feasible partial solutions to the instance based on the specific local structure. Each such enumeration induces a new reduced problem instance that is associated with a child of the node α in the search tree. The search process is then applied recursively to the children of α. The complexity of a branch-and-search process, which is roughly the size of the search tree, depends mainly on two things: how effectively the feasible partial solutions are enumerated, and how efficiently the instance size is reduced. In particular, all exact algorithms proposed in the literature for the Maximum Independent Set problem and the Vertex Cover problem are based on this strategy, and most improvements were obtained by more effective enumerations of feasible partial solutions and/or more efficient reductions in the size of the problem instance [22, 12, 28, 44].

A desirable local structure may not exist at a stage of the branch-and-search process. In this case, the branch-and-search process has to pick a less favorable local structure and make a less effective branch and/or less efficient instance-size reduction. Most proposed branch-and-search algorithms for NP-hard problems were analyzed based on the worst-case performance. That is, the computational complexity of the algorithm was derived based on the worst local structure occurring in the search process. This worst-case analysis for a branch-and-search process is very conservative — the worst cases can appear very rarely in the entire process, while most other cases permit much better branching and reductions.

In this chapter, we suggest new methods to analyze the branch-and-search process. First of all, we label the nodes of a search tree to record the reduction in the parameter size for each branching process. We then perform an amortized analysis on each path in the search tree. This allows us to capture the following notion: an operation by itself may be very costly in terms of the size of the search tree that it corresponds to, however, this operation might be very beneficial in terms of introducing many efficient branches and reductions in the entire process. Therefore, the expensive operation can be well balanced by the induced efficient operations.

This analysis has also enabled us to consider new algorithm strategies in a branch-and-search process. In particular, now we do not have to always strictly avoid expensive operations. To illustrate our analysis and algorithmic techniques, we propose a very simple branch-and-search algorithm for Vertex Cover on degree-3 graphs, abbreviated VC-3. The algorithm also induces a new algorithm for Maximum Independent Set on degree-3 graphs, abbreviated IS-3. Using the new analysis and algorithmic strategies, we are able to show that the new algorithms improve the best existing algorithms in the literature. More specifically, our algorithm for VC-3 runs in time $O(1.194^k + n)$, improving the previous best algorithm of running time $O(1.237^k + kn)$ [23], and our algorithm for IS-3 runs in time $O(1.1255^n)$, improving the previous best algorithm of running time $O(1.1259^n)$ [45].

We would like to further comment on why we picked VC-3 and IS-3 as our candidates. Vertex Cover and Maximum Independent Set are among the most extensively studied NP-hard problems with many proposed algorithms [22, 36, 12, 24, 46, 47, 28, 48, 27, 44]. In particular, Vertex Cover and Maximum Independent Set on graphs of degrees 3 and 4 have received a lot of attention recently [45, 12, 23]. In spite of the restriction imposed on graph degrees (being bounded by 3 or 4), improvements on the previous upper bounds for these problems can be challenging and meticulous.

Moreover, most of the algorithms for Vertex Cover and Maximum Independent Set on general graphs end up reducing the problem to that on low-degree graphs [12, 47, 28]. Thus, a simple and uniform algorithm that induces significant improvements on the existing bounds for these problems is of high interest, and shows the power and effectiveness of the new analysis and algorithmic methods. In addition, recent research has shown that these problems are "complete" in terms of their worst case running time for a large group of well-known NP-hard problems [14, 15, 16]. More specifically, combining the results in [15], [16], and [14], one can show that if IS-3 can be solved in time $O((1+\epsilon)^n)$, or if VC-3 can be solved in time $(1+\epsilon)^k p(n)$ (p is a polynomial), for every constant $\epsilon > 0$, then k-SAT, Maximum Independent Set, and Vertex Cover can all be solved in subexponential time, which seems very unlikely. Hence, it is believed that there are constants $c_1, c_2 > 0$, such that IS-3 and VC-3 have no exact algorithms of running time $O((1+c_1)^n)$ and $(1+c_2)^k p(n)$, respectively. Thus, further improvement in the base of the exponential function in the running time of the algorithms that solve these problems may lead to better understanding of the problems and their associated complexity class.

B. The Main Algorithm

We will assume, without loss of generality, that the graph G in an instance (G, k) of VC-3 contains no isolated vertices (such vertices can be removed in $O(|G|)$ preprocessing time). The number of edges $|E|$ in G then satisfies $|E| \geq |G|/2$ (note that G may not be connected). The degree of G is bounded by 3, and hence, every vertex in G can cover at most three edges. This means that, in order for a vertex cover of size k to exist in G, k must be at least as large as $|E|/3$ (and hence, $k \geq |G|/6$); otherwise, we can report that the answer to the instance (G, k) is negative.

Proposition II.1 allows us to assume, without loss of generality, that in an in-

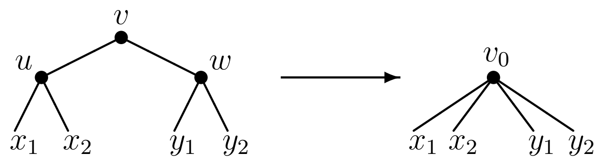

Fig. 4. Vertex folding

stance (G, k) of the VC-3 problem, the graph G contains at most $2k$ vertices.

Let v be a degree-2 vertex in the graph with two neighbors u and w such that u and w are not adjacent. We construct a new graph G' as follows: remove the vertices v, u, and w and introduce a new vertex v_0 that is adjacent to all neighbors of u and w in G (of course except the vertex v). We say that the graph G' is obtained from the graph G by *folding* the vertex v. See Figure 4 for an illustration of this operation. Note that this operation is a spacial case of the *general folding* operation introduced in the previous chapter. We have the following lemma [12], which is a special case of Lemma II.4.

Lemma III.1 ([12]) *Let G' be a graph obtained by folding a degree-2 vertex v in a graph G, where the two neighbors of v are not adjacent to each other. Then $\tau(G) = \tau(G') + 1$.*

Following the terminology of Tutte [49], we define the *binding set* of an induced subgraph H of a graph G to be the set of vertices in H that have neighbors not in H. We first discuss how a small induced subgraph with a small binding set helps identifying vertices that are in a minimum vertex cover.

Lemma III.2 *Let (G, k) be an instance of VC-3 where G has no vertex of degree less than 2. If G has an induced subgraph H with a binding set of at most 2 vertices and*

$4 \leq |H| \leq 50$,[1] *then in constant time we can construct an instance* (G', k') *of VC-3 with a reduced parameter* $k' < k$*, such that* G *has a vertex cover of* k *vertices if and only if* G' *has a vertex cover of* k' *vertices.*

Proof. First we discuss the case where the binding set of H consists of one vertex v. Consider the algorithm **BindingSet1()** in Figure 5. The algorithm runs in constant time since $|H| \leq 50$. If H has a minimum vertex cover containing the vertex v, then let C_H be this vertex cover, otherwise let C_H be any minimum vertex cover of H. In both cases, removing the vertex set C_H from the graph G (and all isolated vertices resulted from this process) gives the graph G'. Thus, it suffices to show that there is a minimum vertex cover C of the graph G that contains the entire set C_H: in this case $C - C_H$ makes a minimum vertex cover for the graph G' and $|C - C_H| = \tau(G) - \tau(H)$.

```
**BindingSet1**$(G, H, v)$
{* $\{v\}$ is the binding set of the induced subgraph $H$ of $G$ *}
**if** $H$ has a min-vc containing the vertex $v$ **then**
    $G' = G - H$;  $k' = k - \tau(H)$;
**else** $G' = G - (H - \{v\})$;  $k' = k - \tau(H)$.
```

Fig. 5. Removing an induced subgraph whose binding set has only one vertex

Let C_G be any minimum vertex cover of G, then $C_G \cap V_H$ is a vertex cover for H, and hence $|C_G \cap V_H| \geq \tau(H)$. If the minimum vertex cover C_H of H contains v, then replacing $C_G \cap V_H$ in C_G by C_H gives a minimum vertex cover for G that contains C_H. On the other hand, suppose C_H does not contain v, i.e., H has no minimum vertex cover containing v. Then in case $v \notin C_G$, replacing $C_G \cap V_H$ in C_G

[1]The constant 50 used here can be replaced by any sufficiently large constant without affecting the correctness of the results in this chapter.

by C_H gives a minimum vertex cover for G that contains C_H; while in case $v \in C_G$, $|C_G \cap V_H| \geq \tau(H) + 1$, and replacing $C_G \cap V_H$ in C_G by C_H plus the vertex v gives a minimum vertex cover of G that contains C_H. Thus, in both cases, there is a minimum vertex cover of G that contains C_H. This proves the lemma for the case where H has a binding set of one vertex.

Now suppose the binding set of H has two vertices u and v. Consider the algorithm **BindingSet2()** in Figure 6, which examines all possible situations in which vertices u and v are contained in minimum vertex covers of H.

```
**BindingSet2**$(G, H, u, v)$
$\{*\ \{u, v\}$ is the binding set of the induced subgraph $H$ of $G\ *\}$
1. **if** $H$ has a min-vc $C_1$ containing both $u$ and $v$
      **then** $G' = G - H$;  $k' = k - \tau(H)$;
2. **else if** $H$ has a min-vc $C_2$ containing $u$ but no min-vc containing $v$
      **then** $G' = G - (H - \{v\})$;  $k' = k - \tau(H)$;
3. **else if** $H$ has a min-vc $C_3$ containing $v$ but no min-vc containing $u$
      **then** $G' = G - (H - \{u\})$;  $k' = k - \tau(H)$;
4. **else if** $H$ has a min-vc $C_4$ containing $u$ and a min-vc $C_4'$ containing $v$
      **then** $G' = G - (H - \{u, v\})$;  $k' = k - \tau(H) + 1$;
          if $[u, v]$ is not an edge, add an edge $[u, v]$ to $G'$;
5. **else** $\{*$ every min-vc of $H$ contains neither $u$ nor $v\ *\}$
      let $\overline{C}_H$ be a smallest vc of $H$ that contains both $u$ and $v$;
      **if** $|\overline{C}_H| = \tau(H) + 2$ **then** $G' = G - (H - \{u, v\})$;  $k' = k - \tau(H)$;
          **else**  $G' = G - (H - \{u, v\})$;  $k' = k - \tau(H) + 1$;
              add a new vertex $w$ and two edges $[w, u]$ and $[w, v]$ to the graph $G'$.
```

Fig. 6. Removing an induced subgraph whose binding set consists of two vertices

For each of the cases 1-3, we only need to verify that the corresponding minimum vertex cover of H is entirely contained in a minimum vertex cover of G. For this, let C_G be any minimum vertex cover of the graph G. In case 1, replacing $C_G \cap V_H$ in C_G by C_1 gives a minimum vertex cover of G that contains C_1. For case 2, if C_G does not contain v, then replacing $C_G \cap V_H$ in C_G by C_2 gives a minimum vertex cover for G; while if C_G contains v then $|C_G \cap V_H| \geq \tau(H) + 1$ (since H has no minimum vertex cover containing v), thus replacing $C_G \cap V_H$ in C_G by C_2 plus v gives a minimum vertex cover of G. The proof for case 3 is completely similar to that for case 2.

Consider case 4. Since $[u, v]$ is an edge in G', every vertex cover of G' must contain at least one of u and v. Moreover, if G has a minimum vertex cover C that contains neither u nor v, then replacing $C \cap V_H$ in C by C_4 gives a minimum vertex cover of G that contains u. Thus, the graph G has a minimum vertex cover C_G that contains at least one of u and v. If C_G contains u but not v, replacing $C_G \cap V_H$ in C_G by C_4 gives a minimum vertex cover C'_G for G satisfying that $(C'_G - C_4) \cup \{u\}$ is a minimum vertex cover for G'. Therefore, $\tau(G') = \tau(G) - |C_4| + 1 = \tau(G) - \tau(H) + 1$. The case in which C_G contains v but not u can be verified similarly using C'_4 instead of C_4. Finally, suppose that C_G contains both u and v. Since case 1 has been excluded and $C_G \cap V_H$ is a vertex cover for H that contains both u and v, we have $|C_G \cap V_H| \geq \tau(H) + 1$. Therefore, replacing $C_G \cap V_H$ in C_G by C_4 plus v gives a minimum vertex cover C''_G for G satisfying that $(C''_G - C_4) \cup \{u\}$ is a minimum vertex cover for the graph G'. Thus again $\tau(G') = \tau(G) - \tau(H) + 1$.

For case 5, let C_H be any minimum vertex cover of H. First consider the subcase $|\overline{C}_H| = \tau(H) + 2$. Let C_G be any minimum vertex cover of G. If C_G contains u but not v, then $|C_G \cap V_H| \geq \tau(H) + 1$ (since no minimum vertex cover of H contains u), so replacing $C_G \cap V_H$ in C_G by C_H plus u gives a minimum vertex cover of G. The case that C_G contains v but not u can be verified similarly. Finally, if C_G contains both u and v, then $|C_G \cap V_H| \geq |\overline{C}_H| = \tau(H) + 2$, and replacing $C_G \cap V_H$ in C_G by C_H plus u and v gives a minimum vertex cover for G. Therefore, in case $|\overline{C}_H| = \tau(H) + 2$, we can simply remove C_H and reduce the parameter by $\tau(H)$. Now consider the subcase $|\overline{C}_H| = \tau(H) + 1$. In this case, the graph G has a minimum vertex cover C_G that either contains both u and v or contains neither: if a minimum vertex cover C of G contains exactly one of u and v, then $|C \cap V_H| \geq \tau(H) + 1$ and replacing $C \cap V_H$ in

C by $\overline{C}_H$ gives a minimum vertex cover of G containing both u and v. Moreover, because of the new degree-2 vertex w, the graph G' has a minimum vertex cover C that either contains both u and v or contains neither. If C_G contains both u and v, replacing $C_G \cap V_H$ in C_G by $\overline{C}_H$ gives a minimum vertex cover C'_G of G satisfying that $(C'_G - \overline{C}_H) \cup \{u, v\}$ is a minimum vertex cover for G' of size $\tau(G) - \tau(H) + 1$, while in case C_G contains neither u nor v, the set $(C_G - C_G \cap V_H) \cup \{w\}$ is a minimum vertex cover for G' of size $\tau(G) - \tau(H) + 1$.

Finally, we note that since $|H| \geq 4$ and G has no vertex of degree less than 2, we have $\tau(H) \geq 2$. Therefore, in all cases we have $k' < k$. □

We note that the condition that the vertex degree is bounded by 3 is not used in the proof of Lemma III.2. Therefore, the lemma remains valid for instances of the general Vertex Cover problem.

Before we present our main algorithm, we introduce some definitions and terminologies.

Definition III.3 Let G be a graph in which no vertex has degree larger than 3.

1. A vertex folding operation is *safe* if it does not create vertices of degree larger than 3.
2. A cycle of length l in G is an *alternating cycle* if it contains exactly $\lfloor l/2 \rfloor$ degree-2 vertices of which no two are adjacent.
3. An *alternating tree* T in G is a tree that is an induced subgraph in G such that all degree-1 vertices in T are of degree 3 in G and no two adjacent vertices in T are of the same degree in G. An alternating tree T is *maximal* if no alternating tree contains T as a proper subgraph.

Our main algorithm is a branch-and-search process, given in Figure 7. Each stage of the algorithm starts with an instance (G, k) of VC-3, and tries to reduce the parameter k by identifying a set S of vertices that are entirely contained in a minimum vertex cover of G, and including the vertex set S in the objective minimum vertex cover for G, which will be called *the partial cover* for G, then recursively works on the reduced instances. The subroutine **Fold**(v) simply applies the safe folding operation to a degree-2 vertex v. We also implicitly assume that after each step, the algorithm calls a subroutine **Clean**, which eliminates all isolated vertices and degree-1 vertices (a degree-1 vertex is eliminated by including its neighbor in the partial cover), and updates the graph G, the partial cover, and the parameter k accordingly. In particular, we will assume that at the beginning of each step, the graph contains no vertices of degree less than 2.

```
VC3-solver

Input: an instance $(G, k)$ of VC-3
Output: a vertex cover $C$ of $G$ of size bounded by $k$ in case it exists
1. while Reducing is applicable do apply Reducing;
2. if there is a maximal alternating tree $T$ of at least 4 vertices in $G$
   then branch on the vertices in $T$ that are of degree 3 in $G$;
3. else if there is a degree-2 vertex $v$ then branch on the two neighbors of $v$;
4. else branch on a degree-3 vertex $v$.

Reducing
A. while there exists a degree-2 vertex $v$ such that folding $v$ is safe do Fold($v$);
B. if $G$ has a component $H$ with $|H| \leq 50$ then include a min-vc of $H$ in the cover;
C. else if there are two adjacent triangles $(u, v, w)$ and $(u, v, z)$
   then include $v$ in the cover;
D. else if there is an alternating cycle $K$ in $G$
   then include all degree-3 vertices on $K$ in the cover;
E. else if $G$ has an induced subgraph $H$ with a binding set of at most two vertices, and such that $4 \leq |H| \leq 50$
   then call the subroutine BindingSet1() or BindingSet2().
```

Fig. 7. The algorithm VC3-solver

We explain how each step in the subroutine **Reducing** is done efficiently. The conditions in step A and step C can be verified by checking each degree-2 vertex and each edge in the graph G, respectively. The conditions in step B can be verified by partitioning the graph G into connected components. The conditions in step E can be checked in linear time using the following procedure. First we apply a linear time algorithm (see [50], section 5.3) to the graph G, which identifies all cut-points and constructs all 2-connected components of G. By examining each 2-connected component, we can check if there is any induced subgraph with a binding set of a single vertex that satisfies the conditions in step E. Similarly, applying the linear time algorithm in [51] to the graph G identifies all cut-pairs, and constructs all the 3-connected components in G. By examining each 3-connected component, we can find out if there is any induced subgraph with a binding set of two vertices that satisfies the conditions in step E. To check the conditions for step D, we run the following subroutine: first remove from G the set T of all edges whose two ends are of degree 3 in G and "smooth" each degree-2 vertex v in G by removing v and adding a new edge connecting its two neighbors. Let the resulting graph be G'. Now every alternating cycle C in the original graph G corresponds either to a cycle in G' (in this case C is of even length) or to an edge $[u, v]$ in T where u and v belong to the same connected component of G' (in this case, C is of odd length). Since the number of edges in G is bounded by $O(k)$, all these conditions can be verified in time $O(k)$.

An explanation for step 2 of the algorithm **VC3-solver** is needed. Because of step 1, there is no alternating cycle in the graph G. Since an alternating tree of at least 4 vertices contains at least one degree-3 vertex in G that is of degree larger than 1 in the tree, we can check each degree-3 vertex in G that has at least two degree-2 neighbors. A simple breadth-first-search style construction from such a degree-3 vertex will give a maximal alternating tree in linear time.

Theorem III.4 *The algorithm* **VC3-solver** *solves the VC-3 problem correctly.*

Proof. We first discuss the subroutine **Reducing**. The correctness of step B is obvious, and the correctness of step A and step E is given by Lemma IV.5 and Lemma III.2, respectively. For step C, since every minimum vertex cover of G must contain at least one of u and v, by the symmetry in the structure, we can simply include v. Finally, consider step D. Let W be the set of all degree-3 vertices in the alternating cycle K, $|W| = \lceil l/2 \rceil$, where l is the length of K. Since every minimum vertex cover C_G of G contains at least $\lceil l/2 \rceil$ vertices in K, replacing $C_G \cap K$ in C_G by W gives a minimum vertex cover containing W. This verifies the correctness of step D.

What remains is to verify the correctness of each step in the main algorithm **VC3-solver**. For this, we show that, in each of the branching steps 2–4, at least one of the outcomes of the branching includes only vertices in a minimum vertex cover of the current graph, into the partial cover.

In step 4 we branch at a degree-3 vertex v by either including v in the cover, or excluding it and including all its neighbors. This step is correct since for any vertex v in the graph, it is true that any minimum vertex cover either contains v, or does not contain v and contains all its neighbors. For step 3, let u and w be the two neighbors of the vertex v. Each minimum vertex cover C_G of G contains at most two of v, u, and w. If C_G contains only one of v, u, and w, then the vertex in C_G must be v so both u and w are not in C_G. If C_G contains two of v, u, w, we can always replace these two vertices in C_G by u and w to get a minimum vertex cover of G that contains both u and w. This verifies the correctness of step 3. Finally, consider case 2. Let W be the set of all degree-3 vertices in the alternating tree T. Suppose that G has a minimum vertex cover C_G that contains some vertex v in W but not the entire W. Let N_i be the set of vertices in T such that, for each vertex u in N_i, the unique path from v to u in T has length i. By the definition of an alternating tree, all vertices in N_i are of degree 2 in G if i is odd and of degree 3 in G if i is even. Since v is in the minimum vertex cover C_G, removing v makes all vertices in N_1 become of degree 1. By the observation given earlier, we can safely include all vertices in N_2 in the minimum vertex cover. Now removing all vertices in N_2 makes all vertices in N_3 become of degree 1, so we can include all vertices in N_4 in the minimum vertex cover, and so on. This process will eventually include all vertices in W in the minimum vertex cover, and give a minimum vertex cover of G that contains the entire set W. This verifies that there is a minimum vertex cover of G that either contains the entire set W or contains no vertex in W, and proves the correctness of step 2. $\square$

The main goal of this chapter is to show that the number of leaves in the search tree of the algorithm **VC3-solver** on an instance (G, k) of VC-3 is $O(1.194^k)$. This will be done in Proposition III.16. We first note that the following conditions can be assumed on the input (G, k) to the algorithm **VC3-solver**.

Assumption III.5 *Let (G, k) be an instance of VC-3. We can assume that when the algorithm* **VC3-solver** *is initially called on the instance (G, k) the following holds true:* (1) *the parameter k passed is not larger than the size of a minimum vertex cover of G; and* (2) *G is connected.*

Suppose first that G is connected. Condition (1) can be justified as follows. We start calling the algorithm on G with $k' = 1, 2, \ldots, k$. The first time the algorithm returns a vertex cover of size k', we stop (note that the vertex cover returned in this case must be a minimum vertex cover). Otherwise, no vertex cover of size bounded by k exists. Each call to the algorithm satisfies condition (1). It will be shown in Proposition III.16 that the number of the leaves in the search tree of the algorithm when called on an instance (G, k) is $O(1.194^k)$. The number of leaves in the search tree in the previous calls to the algorithm becomes bounded by $c \cdot 1.194^1 + c \cdot 1.194^2 +$

$\ldots + c \cdot 1.194^k = O(1.194^k)$ (where c is a positive constant). Hence, the upper bound on the number of leaves in the search tree with the new modification to the algorithm is unchanged. Now to justify (2), suppose that there are $G_1, \ldots, G_r$ components in G with $|G_i| = n_i$. By Proposition II.1, we may assume that the size of a minimum vertex cover of G_i, $\tau(G_i)$, is $\geq n_i/2$. We will also assume that $\tau(G_i) \geq 4$ (a component G_i with $\tau(G_i) < 4$ has its size bounded by 8, and thus can be removed in constant time). We call the algorithm on G_1, with $k_1 = n_1/2, n_1/2+1, \ldots, k$. The first time the algorithm returns a vertex cover of size k_1, we stop. If the algorithm fails to return a vertex cover in each of these cases, then no vertex cover of size bounded by k exists. Otherwise, the algorithm returns a minimum vertex cover of G_1 of size $4 \leq k_1 \leq k$. Now we call the algorithm on G_2 with $k_2 = n_2/2, n_2/2+1, \ldots, k-k_1$, and so on. It is now true that on each call to the algorithm on a graph component, conditions (1) and (2) hold true. The number of leaves in the search tree is $O(1.194^{k_1} + \cdots + 1.194^{k_r})$. We show next that $1.194^{k_1} + \cdots + 1.194^{k_r} \leq 1.194^{k_1+\cdots+k_r}$, which gives that the number of leaves in the search tree is $O(1.194^{k_1+\cdots+k_r}) = O(1.194^k)$.

Since $k_i \geq 4$ for all i, we have $1.194^{k_i} \geq 2$. For any two numbers $a \geq 2$ and $b \geq 2$, we have $ab - (a+b) = (a-1)(b-1) - 1 \geq 0$, which gives $a + b \leq ab$. Using this inequality repeatedly gives

$$\begin{aligned} 1.194^{k_1} + 1.194^{k_2} + 1.194^{k_3} + \cdots + 1.194^{k_r} &\leq 1.194^{k_1+k_2} + 1.194^{k_3} + \cdots + 1.194^{k_r} \\ &\leq 1.194^{k_1+k_2+k_3} + \cdots + 1.194^{k_r} \\ &\leq \cdots \leq 1.194^{k_1+k_2+k_3+\cdots+k_r}. \end{aligned}$$

C. Analysis of the Algorithm

We analyze the time complexity of the algorithm **VC3-solver** in this section. Let $\mathcal{T}$ be the search tree for the algorithm **VC3-solver** on the input instance (G, k). Let α

be a node in the search tree with an associated parameter k'. If we perform a two-sided branch at α by reducing the parameter k' in each branch by a and b, respectively, then such a branch will be called an (a, b) *branch*. We will always assume that in an (a, b) branch, we have $a \leq b$. We say that an (a, b) branch is *not worse than* an (a', b') branch if $a \geq a'$ and $b \geq b'$.

Differing from the common analysis techniques based on the worst-case scenario, we present next a novel way for analyzing the size of the search tree. This can be achieved by looking at the set of operations performed by the algorithm as an interleaved set of operations. This allows us to counter-balance the effect of inefficient operations with efficient ones, thus providing a better upper bound on the size of the search tree. Our goal is to show that the number of leaves in the search tree $\mathcal{T}$ is $O(r^k)$, where $r \leq 1.194$ is the unique positive root of the polynomial $x^k - x^{k-3} - x^{k-5}$, or equivalently, the unique positive root of the polynomial $x^5 - x^2 - 1$.

The graph G is called *clean* if no vertex of degree 0 or 1 exists in G. The graph G is called *nice* if it is clean and no safe folding is applicable to any vertex in G. We will divide the operations performed by the algorithm into four categories.

1. **Folding** operations: the operations performed in step A of the subroutine **Reducing**.

2. $(1, 3)$ **branching** operations: the operations performed in step 4 of **VC3-solver** when we branch on a degree-3 vertex. These operations occur only when the graph becomes 3-regular.

3. $(2, 5)$ **branching** operations: the operations performed in step 3 of **VC3-solver** when we pick a degree-2 vertex and branch on its neighbors. Note that at this point of the algorithm the graph is nice, and hence, no safe folding is applicable. Also, step D of **Reducing** is not applicable. This means that the two vertices

that we branch on have five neighbors, and the branch in this case is a $(2, 5)$-branch.

4. The operations performed in: steps B-E of **Reducing**, step 2 of **VC3-solver**, and those performed by the subroutine **Clean**.

The operations will be referred to by their categories. For example, a category-1 operation denotes a folding operation, and a category-4 operation denotes one of the operations listed in number 4 above.

Let i be an operation[2] in any of the above categories. We define the following parameters for operation i: e_i the number of edges removed in operation i, v_i the number of vertices removed in operation i, and k_i the reduction in the parameter by operation i. We define the *surplus* s_i of operation i as follows. If i is a non-branching operation that reduces the parameter by k_i, then $s_i = k_i$. If i is the a-side (resp. b-side) of a branching operation (a, b), where $a \leq b$, then $s_i = a - 3$ (resp. $s_i = b - 5$). Informally speaking, s_i is the addition or reduction in the parameter, relative to a $(3, 5)$-branch, that is gained or lost in the operation i. We define the *amortized cost* m_i of operation i by $m_i = 5e_i - 6v_i + 6s_i - 3k_i$. Note that if the operation i is followed by **Clean**, we will combine the amortized cost of **Clean** with m_i. Also note that for any non-branching operation $s_i = k_i$, therefore the amortized cost of such an operation is $m_i = 5e_i - 6v_i + 3k_i$.

The amortized cost m_i defined above will be used to measure the cost related to operation i including the benefit cost generated by operation i, the cost gained by operation i from other previous operations, and the cost relative to attaining the

[2]When looking at the search tree, a branching operation will denote the two sides of the branch, whereas when looking at a certain path in the search tree, one side of a branching operation will be considered an operation by itself. It should be clear from the context what is meant by a branching operation (i.e., either one side of the branch or the whole branch).

target parameter reduction of the operation. Based on the principle of "gain more then pay more", we use the gain in the parameter reduction related to the operation to measure the corresponding cost. Write $s_i = k_i - \delta_i$, where δ_i is the "target value" in the parameter reduction for operation i (e.g., for an (a, b) branch, where $a \leq b$, the target value for the a-side operation is 3, and for the b-side operation is 5). Rewrite the formula as $m_i = (5e_i - 6v_i) + 3s_i - 3\delta_i$. We consider the three parts in the formula for the amortized cost m_i. (A) The term $(5e_i - 6v_i)$ in m_i: observe that for a clean graph of n vertices and m edges, if the edge/vertex ratio m/n is less than 6/5, then a safe folding operation is applicable (see Proposition III.18). Thus, if the operation i removes e_i edges and v_i vertices such that $e_i/v_i > 6/5$ (or, equivalently $5e_i - 6v_i > 0$), then the operation i will lower the edge/vertex ratio in the remaining subgraph and increase the possibility of safe folding, which will benefit later steps of the algorithm. Therefore, the term $(5e_i - 6v_i)$ in m_i describes the cost of the operation i that will benefit later steps of the algorithm. (B) The surplus s_i: the value of s_i represents the gain in the parameter reduction that is beyond the target value. Note that in the algorithm, each operation i with a positive surplus must have taken the advantage of a certain special graph structure which had been generated by previous operations. Moreover, after the operation i, the favored structure disappears. Therefore, the value s_i can be regarded as the cost of previous operations to generate the favored structure consumed by the operation i. For example, a safe folding operation takes the advantage of two adjacent degree-2 vertices (which are generated by previous operations), gains a surplus 1, but eliminates the favored structure (i.e., the two adjacent degree-2 vertices). Therefore, the value s_i describes the cost of previous operations that benefited the operation i. (C) The value δ_i: since the cost of the operation i spent for gaining the target parameter reduction δ_i is excluded from the amortized cost, the term $-\delta_i$ becomes a term in the amortized cost m_i.

Based on the above discussion, it is natural to define the amortized cost as a linear function of $(5e_i - 6v_i)$, s_i, and $-\delta_i$. The remaining question is to determine the coefficients of these terms, i.e., to determine how these terms are proportionally related. We give an intuitive explanation here. The entity s_i counts the extra reduction in the parameter value, and the entity δ_i denotes the targeted reduction in the parameter value. Therefore, both s_i and δ_i refer to the reduction in the parameter value, and hence, it makes sense to give them the same coefficient in the formula for m_i. Now how is the term $(5e_i - 6v_i)$ related to the value s_i (and δ_i)? A careful analysis of the algorithm (see the proofs of Proposition III.17 and Lemma III.19) shows that it is proper to equate a value 3 in $(5e_i - 6v_i)$ to a value 1 in s_i. We use the 1-side operation of a $(1, 3)$ branch as an example. Here we have $e_i = 3$ and $v_i = 1$, thus $(5e_i - 6v_i) = 9$. On the other hand, the operation creates three degree-2 vertices, each may induce a folding that reduces the parameter by 1. Therefore, a value 9 in the term $(5e_i - 6v_i)$ seems to correspond to a value 3 in the parameter reduction. The same conclusion can be derived for the 2-side operation of a $(2, 5)$ branch.

The above explains the main intuition behind the formulation of the amortized cost as $m_i = (5e_i - 6v_i) + 3s_i - 3\delta_i$, which is equivalent to $m_i = 5e_i - 6v_i + 6s_i - 3k_i$.

Lemma III.6 *Let C_0 be a connected component in G, and let m_0 be the amortized cost incurred by invoking* **Clean** *on C_0. If C_0 is not a tree then $m_0 \geq 0$, and if C_0 is a tree then $m_0 \geq -6$.*

PROOF. Suppose first that C_0 is a non-tree connected component in G. Let e_0, v_0, k_0 be the parameters of the operation of applying **Clean** to C_0. Since **Clean** is a non-branching operation, we have $m_0 = 5e_0 - 6v_0 + 3k_0$. If **Clean** removes the whole component C_0, then since C_0 is connected and is not a tree, we have $e_0 \geq v_0$. Also, $k_0 \geq e_0/3$ since every removed edge must be covered by the vertices that have been included in the vertex cover, and each vertex can cover at most 3 edges. It follows that the amortized cost $m_0 = 5e_0 - 6v_0 + 3k_0 \geq 0$. Now suppose that **Clean** does not remove the whole component C_0. Then any connected induced subgraph C' of C_0 that is removed by **Clean** must have at least one edge connecting it to $V(C_0) - V(C')$, which is also removed by **Clean**. It follows that the number of edges e' removed when removing C' is at least as large as the number of vertices v' in C'. Also, the reduction in the parameter k' incurred in C' is $k' \geq e'/3$ by the same argument as above. It follows that the amortized cost m' induced by m_0 on every connected subgraph C' of C removed by **Clean** is non-negative. The amortized cost m_0 on C_0 is the summation of the amortized cost on each connected subgraph removed by **Clean** (this follows from the linearity of the expression for the amortized cost and the monotonicity of addition). It follows that the amortized cost m_0 incurred by cleaning a non-tree component is always non-negative.

Suppose now that C_0 is a tree. In this case **Clean** removes the whole component C_0. It follows that $e_0 = v_0 - 1$. This, together with $k_0 \geq e_0/3$, gives $m_0 = 5e_0 - 6v_0 + 3k_0 \geq -6$. □

Lemma III.7 *A non-branching operation on a connected component of a clean graph G has a non-negative amortized cost.*

PROOF. Since G is clean, every connected component of G is also clean, and hence, is not a tree. It follows, by a similar argument to that in Lemma III.6, that the induced amortized cost on every connected subgraph of G removed by the operation plus **Clean** is non-negative. Hence, the total amortized cost is non-negative. □

Fact III.8 *A tree with exactly two degree-1 vertices is a path between the two degree-1 vertices.*

Lemma III.9 *On a nice graph G, an operation i performed in step E of* **Reducing** *followed by an invocation to* **Clean** *has a non-negative amortized cost m_i. In particular, the amortized cost of step 4 of the procedure* **BindingSet2()** *is at least 6.*

PROOF. In step E of **Reducing**, the algorithm removes a subgraph from G and possibly adds some edges and vertices to the graph. We need to verify that the amortized cost of such an operation is non-negative. In the cases when the operation does not add any vertices or edges to the graph, the fact that the amortized cost is non-negative follows from Lemma III.7. We only need to show this statement for step 4 of **BindingSet2()** when one edge is added, and step 5 of **BindingSet2()**, when one vertex and two edges are added. We show the statement for step 4 of **BindingSet2()**. The proof that this statement holds true for step 5 of **BindingSet2()** is very similar. The operation in step 4 removes $(H - \{u, v\})$ from G and adds an edge $[u, v]$ if this edge does not already exist. If the edge $[u, v]$ already exists, then no edge is added and we are done. Suppose that there is no edge $[u, v]$ in G. Note that H cannot be a tree, otherwise, since the operation is performed on a clean connected component of the graph, H would have exactly two degree-1 vertices namely u and v, and by Fact III.8, H must be a chain (note that a tree with more than one vertex must have at least two degree-1 vertices). Since $|H| \geq 4$, this would imply that there were two adjacent degree-2 vertices in the graph prior to this operation contradicting the fact that no safe folding is applicable at this stage of the algorithm. Thus, we must have $e_H \geq v_H$, where e_H and v_H are the number of edges and vertices in H, respectively. The operation removes $e_H - 1$ edges (e_H edges from H, and $[u, v]$ is added), $v_H - 2$ vertices, and reduces the parameter by k_H. Since each of the k_H vertices included in the vertex cover can cover at most 3 edges, we must have $k_H \geq (e_H - 1)/3$. Since the operation is a non-branching operation, its amortized cost $m_i = 5(e_H - 1) - 6(v_H - 2) + 3k_H \geq 6e_H - 6v_H + 6 \geq 6$. Also, since prior to this operation the graph was clean, the resulting graph is also clean, and hence, the subroutine **Clean** is not applicable. This completes the proof. $\square$

Proposition III.10 *Let G be a nice graph, and let $\mathcal{S}$ be a collection of disjoint induced trees in G that are joined to $G - \mathcal{S}$ by l edges. Then $|\mathcal{S}| \leq 4l - 7$.*

PROOF. It suffices to prove the proposition for the case when $\mathcal{S}$ contains a single induced tree T. The proof for the general case follows by applying the statement to each induced tree in $\mathcal{S}$.

For an induced tree T, let L_T be the set of vertices of degree less than 2 in the tree T, and let C_T be the set of edges with one end in T and the other end in $G - T$. We prove, by induction on $|T|$, the following statement:

> **Statement A.** $|T| \leq 4|C_T| - 7$. More precisely, if a vertex in L_T has degree 3 in the graph G, then $|T| \leq 4|C_T| - 10$, and if all vertices in L_T have degree less than 3 in G, then $|T| \leq 4|C_T| - 7$.

First note that the graph G is nice, and hence, G has no vertices of degree less than 2. When $|T| = 1$, if the vertex v in T has degree 3 in G then $|C_T| = 3$, and if the vertex v has degree 2 in G then $|C_T| = 2$. Therefore, **Statement A** holds true when $|T| = 1$. When $|T| = 2$, T consists of a single edge $[u, w]$, and $|C_T| \geq 3$, since the nice graph G cannot have two adjacent degree-2 vertices u and w. Therefore, **Statement A** holds true when $|T| = 2$.

Now consider the general case $|T| \geq 3$. First suppose that there is a vertex w in L_T such that w is of degree 3 in G. Then one edge $[w, u]$ incident on w is in T (because $|T| > 1$), and the other two edges $[w, w_1]$ and $[w, w_2]$ incident on w belong to C_T. Consider the tree $T' = T - \{w\}$ in G. We have $|T'| = |T| - 1$ and $|C_{T'}| = |C_T| - 1$

($C_{T'}$ is obtained from C_T by removing the two edges $[w, w_1]$ and $[w, w_2]$ and adding the edge $[w, u]$). By the inductive hypothesis, $|T'| \leq 4|C_{T'}| - 7$, which gives directly that $|T| \leq 4|C_T| - 10$.

Now suppose that all vertices in L_T have degree 2 in G. Pick a longest path in T with endpoints r and w. Both r and w must be in L_T, and hence, have degree 2 in G. Let u be the neighbor of w in the tree T (the vertex u must exist, and must be different from r, because $|T| \geq 3$ and a longest path in T from r to w has length at least 2).

Let the other edge incident on w be $[w, w_1]$. Since the graph G is nice, the vertex u must be of degree 3 in G (otherwise, w and u would be two adjacent degree-2 vertices in G). Let the edge incident on u but not on the path joining r to w be $[u, u_1]$. If u_1 is not in T, then consider the tree $T' = T - \{w\}$, and note that u is in $L_{T'}$. We have $|T'| = |T| - 1$, and $|C_{T'}| = |C_T|$ ($C_{T'}$ is obtained from C_T by removing the edge $[w, w_1]$ and adding the edge $[u, w]$). Since u is of degree 3 in G, by the inductive hypothesis, we have $|T'| \leq 4|C_{T'}| - 10$, which gives $|T| \leq 4|C_T| - 9 < 4|C_T| - 7$. Suppose now that u_1 is in the tree T. Then u_1 must be in L_T (otherwise, the path from r to w would not be a longest path in T), and u_1 has degree 2 in G. Consider the tree $T'' = T - \{w, u_1\}$ in G. We have $|T''| = |T| - 2$, and $|C_{T''}| = |C_T|$ ($C_{T''}$ is obtained from C_T by removing the edge $[w, w_1]$ and the edge joining u_1 to $G - T$, and adding two edges $[u, w]$ and $[u, u_1]$). Now the vertex u is in $L_{T''}$, and u is of degree 3 in G. By the inductive hypothesis, $|T''| \leq 4|C_{T''}| - 10$, which gives $|T| \leq 4|C_T| - 8 < 4|C_T| - 7$.

This completes the inductive proof of **Statement A** and the proof of the proposition. □

Lemma III.11 *On a nice graph G, an operation i performed in step 2 of* **VC3-solver** *followed by an invocation to* **Clean** *is not worse than a $(3, 5)$-branch, and its amortized cost m_i is non-negative.*

Proof. We first prove a general result for alternating trees. Suppose that T is an alternating tree with at least 3 vertices. Let D_2 and D_3 be the sets of vertices in T of degree 2 and degree 3 in G, respectively, and let $x = |D_3|$. Let Y be the set of neighbors of D_3 that are not in T, i.e., $Y = N(D_3) - D_2$, and let $y = |Y|$. We first show, by induction on $|T|$, that (1) $|D_2| = x - 1$ and hence $|T| = 2x - 1$; and (2) there are exactly $(x + 2)$ edges between T and Y.

For the base case $|T| = 3$, from the definition of an alternating tree, the tree T must be a chain $[u_1, u_2, u_3]$ of three vertices, where u_1 and u_3 are of degree 1 in T and degree 3 in G, and u_2 is of degree 2 in both T and G. Moreover, there are four edges joining T to $G - T$, namely those edges joining u_1 and u_3 to the vertices in $G - T$. Therefore, we have $x = |D_3| = 2$, $|D_2| = 1$, and the number of edges between T and Y is 4. Thus, statements (1) and (2) hold true in this case.

We note that the case $|T| = 4$ is impossible: if T has three degree-1 vertices (which are of degree 3 in G), then the fourth vertex in T must be connected to all the three degree-1 vertices, and hence cannot be of degree 2, so T is not an alternating tree; while if T has two degree-1 vertices, then the other two vertices in T must be of degree 2 and adjacent, so again T would not be an alternating tree.

Therefore, for a general case for an alternating tree T with $|T| > 3$, we must have $|T| \geq 5$. Let w be any vertex of degree 1 in T. By the definition of alternating trees, w is of degree 3 in the graph G. The vertex w is adjacent to a degree-2 vertex u in the tree T and adjacent to two other vertices w_1 and w_2 in $G - T$. Let the other neighbor of u in T be u_1, which is a degree-3 vertex in G. Consider the tree $T' = T - \{w, u\}$ in G. Then $|T'| = |T| - 2 \geq 3$. Moreover, the tree T' is an alternating tree: the degree-3 vertex u_1 now becomes of degree 1 in T', and the degrees of the vertices in T' still alternate. Let D_2' and D_3' be the sets of vertices in T' of degree 2 and degree 3 in G, respectively. Then $|D'_2| = |D_2| - 1$ and $|D'_3| = |D_3| - 1$. Moreover, the number of edges β' between T' and $G - T'$ is exactly one less than the number of edges β between T and $G - T$ (the set of edges between T' and $G - T'$ is obtained from the set of edges between T and $G - T$ by removing the two edges $[w, w_1]$ and $[w, w_2]$ and adding the edge $[u_1, u]$). By the inductive hypothesis, we have $|D'_2| = |D'_3| - 1$ and $\beta' = |D'_3| + 2$, which gives directly that $|D_2| = |D_3| - 1 = x - 1$ and $\beta = |D_3| + 2 = x + 2$. This completes the proof of statements (1) and (2).

Now we are ready to prove the statement of the lemma. Since the number of vertices in an alternating tree is $2x - 1$, which is an odd number, and since $|T|$ is assumed to be ≥ 4 in step 2 of **VC3-solver**, we have $|T| \geq 5$, and hence, $x \geq 3$. Part (2), and the fact that $x \geq 3$, imply that there are at least five edges between T and Y. Since every vertex in the graph has degree bounded by 3, we have $y \geq 2$.

If $y = 2$, then $x \leq 4$, and the subgraph H induced by $V(T) \cup Y$ has size at most 9. Since no isolated components of size ≤ 50 exist at this point of the algorithm by step B in **Reducing**, the binding set of H has size bounded by 2 (the binding set of H is a subset of Y). Since $4 \leq |H| \leq 50$, this is not possible at this point of the algorithm by step E of **Reducing**. It follows that $y \geq 3$, and branching in step 2 of **VC3-solver** on D_3 gives a $(|D_3|, |D_2| + |Y|) = (x, x - 1 + y)$ branch, which is not worse than a $(3, 5)$-branch since both x and y are at least 3.

What is left is showing that the amortized cost m_i of operation i is non-negative. Consider first the side of the branch where we include the vertices in D_3 in the partial cover. The vertices removed by this branch are those in T whose number is $v_i = 2x - 1$. The edges removed are those in T plus the edges between T and Y. These edges are exactly the edges incident on the vertices in D_3. Since no two degree-3 vertices in T are adjacent, it follows that the number of edges e_i removed by the branch is $3x$. Moreover, the reduction k_i in the parameter is x, and the surplus is $x - 3$. Now let $\mathcal{S}$

be the set of tree components in the resulting graph $G - T$, and let t_i be the number of tree components in $\mathcal{S}$. By Lemma III.6, the amortized cost of **Clean** on a non-tree component is non-negative, and on a tree component is at least -6. It follows that the amortized cost of operation i including the invocation of **Clean** is

$$\begin{aligned} m_i &\geq 5e_i - 6v_i + 6s_i - 3k_i - 6t_i \\ &\geq 5(3x) - 6(2x-1) + 6(x-3) - 3x - 6t_i \\ &= 6x - 12 - 6t_i. \end{aligned} \tag{3.1}$$

Observe that the tree components in $\mathcal{S}$ are disjoint, and each tree component must be connected by at least two edges to T (since no degree-1 vertices exist in G). It follows from this observation that there cannot be more than $\lfloor (x+2)/2 \rfloor$ tree components in $\mathcal{S}$, and hence, $t_i \leq \lfloor (x+2)/2 \rfloor$. If $x \geq 6$, then from Inequality (3.1), we get $m_i \geq 0$. Suppose now that $x \leq 5$. We claim that in this case either there exists a non-tree component in $G - T$ that is joined to T by at least three edges, or there exist at least two non-tree components in $G - T$. If all components in $G - T$ are tree components, i.e., $G - T = \mathcal{S}$, then $\mathcal{S}$ is a collection of disjoint induced trees that are joined to T by at most $x + 2 \leq 7$ edges satisfying the conditions of Proposition III.10 with $l = 7$. It follows in this case that the number of vertices in $\mathcal{S}$ is bounded by 21, and hence, the total number of vertices in the graph component induced by $V(T) \cup V(\mathcal{S})$ is bounded by 30. This is not possible at this point of the algorithm due to the fact that step B in **Reducing** was not applicable. Now suppose that there is exactly one non-tree component C_0 in $G - T$ that is joined by exactly two edges to T. By a similar argument to the above, the graph induced by $V(T) \cup V(\mathcal{S})$ has at most 22 vertices (and at least 4 vertices), and is connected to C_0 by exactly two edges, which means that it has a binding set of size at most 2. This is again not possible by step E of **Reducing**. It follows that the claim holds true. An immediate consequence of this claim is that $t_i \leq \lfloor (x+2-3)/2 \rfloor = \lfloor (x-1)/2 \rfloor$. Combining this with (3.1), we get $m_i \geq 3x - 9 \geq 0$ because $x \geq 3$.

Now on the other side of the branch we include the neighbors of D_3: D_2 and Y. Let e_Y be the number of edges connecting the vertices of Y, and z the number of edges between the graph induced by $V(T) \cup Y$ and the remaining graph. It is not difficult to verify that in this side of the branch the number of edges e_i removed is $3x + z + e_Y$, the number of vertices v_i removed is $2x - 1 + y$, and the reduction in the parameter k_i is $x - 1 + y$. Let $\mathcal{S}$ be the set of tree components in $(G - T) - Y$, and t_i the number of tree components in $\mathcal{S}$. Now

$$\begin{aligned} m_i &\geq 5e_i - 6v_i + 6s_i - 3k_i - 6t_i \\ &\geq 5(3x + z + e_Y) - 6(2x - 1 + y) + 6(x - 1 + y - 5) - 3(x - 1 + y) - 6t_i \\ &\geq 6x - 3y + 5e_Y + 5z - 6t_i - 27. \end{aligned} \tag{3.2}$$

Since the alternating tree is maximal, all vertices in Y have degree 3. By counting the sum of the degrees of the vertices in Y, we get

$$3y = x + 2 + z + 2e_Y. \tag{3.3}$$

Combining (3.2) and (3.3) and noting that $t_i \leq \lfloor z/2 \rfloor$, we get

$$m_i \geq 5x + 3e_Y + 4z - 6t_i - 29 \tag{3.4}$$

$$\geq 5x + z + 3e_Y - 29. \tag{3.5}$$

If $x \geq 6$, then from Inequality (3.5) we have $m_i \geq 0$. If $x = 5$, then from Inequality (3.5), the fact that $z \geq 3$ (note that if $z \leq 2$ then the graph induced by $V(T) \cup Y$ has size bounded by 50 and a binding set of size at most 2), Equality (3.3), and the fact that y is an integer, we have $m_i \geq 0$. If $x = 4$ and $z \geq 9$, then again by Inequality (3.5), $m_i \geq 0$. We are left with the cases $x = 4$ and $z < 9$, or $x = 3$. If $x = 3$, then $z \leq 10$, because there cannot be more than 5 vertices in Y each of which has to be joined by at least one edge to T. It follows that in both cases $z \leq 10$ and $|V(T) \cup Y \cup V(\mathcal{S})| \leq 50$ (since $|\mathcal{S}| \leq 33$ by Proposition III.10). By an argument similar to the above, we must have at least two non-tree components in $G - (V(T) \cup Y)$, or a non-tree component that is joined to Y by at least three edges. It follows that $t_i \leq \lfloor (z-3)/2 \rfloor$. Combining this with Inequality (3.4), we get

$$\begin{aligned} m_i &\geq 5x + 3e_Y + 4z - 6\lfloor (z-3)/2 \rfloor - 29 && (3.6)\\ &\geq 5x + z + 3e_Y - 20. && (3.7) \end{aligned}$$

Since $x \geq 3$ and $z \geq 3$, if $x = 4$, $z \geq 5$, or $e_Y \geq 2$, by (3.7) we get $m_i \geq 0$. Assume now that $x = 3$, $z \in \{3, 4\}$, and $e_Y \in \{0, 1\}$. Because $x,\ y,\ z,\ e_Y$ are all integers, it is easy to see from (3.3), that the only possible case is when $x = 3,\ y = 3,\ z = 4,\ e_Y = 0$. Substituting these values in (3.6), we get $m_i \geq 2$.

It follows that branch i is not worse than a $(3, 5)$-branch, and the amortized cost of i including the invocation to **Clean** is non-negative. This completes the proof. $\square$

Theorem III.12 *Let i be an operation performed in one of steps B-E in* **Reducing***, or step 2 in* **VC3-solver** *followed by an invocation to* **Clean***. Then the amortized cost m_i of i is non-negative.*

PROOF. By Lemma III.7, the amortized cost corresponding to any non-branching operation is non-negative. In particular, the amortized cost corresponding to an operation performed in any of steps B-D of **Reducing** is non-negative. Lemma III.9

shows that step E of **Reducing** followed by an invocation to **Clean** has a non-negative amortized cost (note that Lemma III.7 cannot be applied to an operation in step E since such an operation may add edges and vertices to the graph). Lemma III.11 establishes the same facts for step 2 of **VC3-solver**. □

Proposition III.13 *Let O be an operation that removes e_0 edges, v_0 vertices, reduces the parameter by k_0, and has surplus s_0. Let $m_0 = 5e_0 - 6v_0 + 6s_0 - 3k_0$ be the amortized cost of operation O.*

(i) *If O is a category-1 operation then $m_0 \geq 1$.*

(ii) *If O is the 1-side branch in a category-2 operation then $m_0 = -6$.*

(iii) *If O is the 3-side branch in a category-2 operation then $m_0 \geq -6$.*

(iv) *If O is the 2-side branch in a category-3 operation then $m_0 = 0$.*

(v) *If O is the 5-side branch in a category-3 operation then $m_0 \geq 1$.*

(vi) *If O is a category-4 operation, then $m_0 \geq 0$.*

Proof. A folding operation removes at least two edges and two vertices. Hence, $e_0 \geq 2$ and $v_0 = 2$. In both cases we have $s_0 = k_0 = 1$ (since there is no branching). It follows that $m_0 \geq 1$. Now in the 1-side of the $(1, 3)$-branch it is always the case that exactly one vertex and three edges are removed. Since $s_0 = -2$ and $k_0 = 1$, we have $m_0 = -6$. Also, the remaining graph is clean, and **Clean** is not applicable. Similarly for the 2-side of the $(2, 5)$-branch, when we branch on the two neighbors w_1 and w_2 of a degree-2 vertex w, 6 edges and 3 vertices are removed, and no degree-1 vertices are created since all the other neighbors of the two vertices w_1 and w_2 must be of degree 3 (otherwise we would have an alternating tree of size at least 5, which is not possible since step 2 of **VC3-solver** was not applicable). Since $s_0 = -1$ and $k_0 = 2$, we have $m_0 = 0$. In all the above cases, the subroutine **Clean** is not applicable since all the remaining vertices have degrees larger than one. This proves parts $(i), (ii), (iv)$.

To prove part (*iii*), note first that in the 3-side of the $(1, 3)$ branching we have $s_0 = -2$ and $k_0 = 3$. Also, we know that before this operation the graph G is 3-regular. Let u be the degree-3 vertex that we branch on, and let v, w, z be its neighbors. Let H be the graph induced by $\{u, v, w, z\}$. Since **Reducing** does not apply at this point, there cannot be more than one edge among v, w, z (otherwise, we would have two adjacent triangles). Suppose that there exists one edge among v, w, z. This means that there are exactly four edges connecting H to $G - H$. Note that in this case no component in $G - H$ can be a tree, otherwise, using Proposition III.10, the graph induced by the vertices of the tree component plus the vertices of H has size bounded by 50, and is connected to the remaining graph by at most two edges (since the tree component has to be connected to $\{v, w, z\}$ by at least two edges), which is not possible at this stage of the algorithm since steps B-E of **Reducing** do not apply. Thus, we can assume that no component in $G - H$ is a tree, and hence by Lemma III.6, the amortized cost of **Clean** in case it is invoked is non-negative. The number of edges and vertices removed in this case is 8 and 4, respectively, giving $m_0 \geq 5e_0 - 6v_0 - 21 = -5$.

Now suppose that no edge exists among v, w, z, and hence, there are exactly six edges connecting H to $G-H$. By a similar argument to the above, we cannot have two different components in $G - H$ that are trees. Thus, in the worst case, the amortized cost of **Clean** is at least -6 by Lemma III.6. The branch itself removes 9 edges and 4 vertices from the graph. Since the total amortized cost is the sum of the amortized cost of the branch and that of **Clean**, it follows that $m_0 \geq 5e_0 - 6v_0 - 27 = -6$.

Now we look at part (v) which is the 5-side of the $(2, 5)$-branch. Note that in this case we have $s_0 = 0$ and $k_0 = 5$. Let u be the degree-2 vertex that we branch on its two neighbors v and w. Let v_1 and v_2 be the neighbors of v other than u, and w_1 and w_2 be those of w. Observe that since folding is not applicable, v and w

must be of degree 3 and they do not share any neighbors except u. Also, since no alternating tree of size ≥ 5 exists at this point, v_1, v_2, w_1, w_2 must be all of degree 3. Let H be the graph induced by $\{u, v, w, v_1, v_2, w_1, w_2\}$. If there are more than two edges among the vertices $\{v_1, v_2, w_1, w_2\}$, the graph H, which has size bounded by 50, would be connected to $G - H$ by at most two edges, which is not possible at this stage of the algorithm (because no induced subgraph with a binding set of size at most two exists). If the number of edges between $\{v_1, v_2, w_1, w_2\}$ is two, then there are exactly four edges connecting H to $G - H$. By a similar argument to the above, there cannot be any tree component in $G - H$, otherwise, there will be at most two edges connecting H and the tree (having size bounded by 50), to the remaining graph. The number of edges and vertices removed in this case is 12 and 7 giving $m_0 \geq 3$, and the amortized cost of **Clean** is positive (since there is no tree component). Now suppose there is exactly one edge between $\{v_1, v_2, w_1, w_2\}$. In this case the number of edges between H and $G - H$ is exactly six, and the number of edges and vertices removed is 13 and 7. By the same token, there cannot be two tree components in $G - H$, and hence the amortized cost of **Clean** is at least -6 by Lemma III.6. This gives $m_0 \geq 5e_0 - 6v_0 - 21 = 2$. If there are no edges among $\{v_1, v_2, w_1, w_2\}$, then there are exactly eight edges connecting H to $G - H$, and the number of edges and vertices removed is 14 and 7. Again, we cannot have more than two tree components in $G - H$ giving an amortized cost of at least -12 for **Clean**. This gives $m_0 \geq 5e_0 - 6v_0 - 27 = 1$. It follows that in all cases of the branch $m_0 \geq 1$.

To prove part (vi), note that a category-4 operation is either an operation performed in steps B-E of **Reducing** followed by an invocation to **Clean**, an operation performed in step 2 of **VC3-solver** followed by an invocation to **Clean**, or one that is performed in **Clean**. If O is an operation that is performed in steps B-E of **Reducing** or in step 2 of **VC3-solver**, then by Theorem V.18, the amortized cost of O

including the call to **Clean** is non-negative. Now if O is an operation in **Clean** that does not follow an operation in steps B-E of **Reducing** or step 2 of **VC3-solver**, by the above discussion, O must be an operation following a 3-side of a $(1,3)$-branch, or a 5-side of a $(2,5)$-branch (these cover all the cases in which **Clean** is called). By parts (iii) and (v) above, the negative part of the amortized cost of **Clean** was combined with the amortized cost of the operation itself, and the remaining part is positive. This completes the proof. $\square$

Based on Proposition III.13, we give in Table I the parameters for any operation i in the four categories. If operation i is a category-4 operation (or one side of a category-4 operation), then we denote its surplus by s_i, reduction in the parameter by k_i, and amortized cost by m_i. For every operation, a lower bound on its amortized cost is given in the last column of the table.

Table I. The parameters of the operations

Operations		reduction in k	surplus	amortized cost
Folding		1	1	1
$(1,3)$ branching	1-side	1	-2	-6
	3-side	3	-2	-6
$(2,5)$ branching	2-side	2	-1	0
	5-side	5	0	1
Category-4 operation i		k_i	s_i	0

Each non-root node α in a search tree $\mathcal{T}$ for the algorithm **VC3-solver** uniquely *specifies* the operation in the algorithm from the parent of α to α. Therefore, each operation in the algorithm can be uniquely referred to by the corresponding node in the tree $\mathcal{T}$. To simplify the description, we also assume that the root of $\mathcal{T}$ has a "virtual" parent associated with the input (G,k) to the algorithm, and that the root of $\mathcal{T}$ specifies a "dummy" operation whose parameter reduction, surplus, and amortized cost, are all equal to 0. Thus, every node in the search tree (including the root) has a parent. By saying *the operations on a path P* in the search tree $\mathcal{T}$, we will be referring to the operations specified by the nodes on P. The reader should note the distinction between *the operation specified by a node* and *the instance* (G', k') *associated with the node* (i.e, the resulting graph G' and the parameter value k' at the node). In particular, the operation specified by a node is actually the operation applied to the instance associated with the parent of the node.

Definition III.14 In a search tree $\mathcal{T}$ of the algorithm **VC3-solver**, we assign to each node α a *label* whose value is equal to the parameter reduction of the operation specified by the node α. More precisely, if the operation from the parent of a node α in $\mathcal{T}$ to α is the a-side (resp. the b-side) of an (a, b) branch, then the label of α is a (resp. b); if the operation from the parent of α to α is a non-branching operation that reduces the parameter value by c, then the label of α is c. As discussed above, the root of $\mathcal{T}$ specifies a dummy operation whose parameter reduction is 0, and hence, the label of the root is 0.

Let P be a path in a search tree $\mathcal{T}$. Denote by $x_1(P)$ the number of nodes on P with label 1, specifying the 1-side operations of (1, 3) branches. Similarly, denote by $x_3(P)$ and $x_2(P)$ the number of nodes on P with labels 3 and 2, specifying the 3-side operations of (1, 3) branches and the 2-side operation of (2, 5) branches, respectively. Finally, denote by $d(P)$ the sum of the surplus of all other operations (i.e., the operations in categories 1 and 4) on the path.

Definition III.15 Let P be a path in a search tree $\mathcal{T}$ of the algorithm **VC3-solver**. The *surplus* of the path P, denoted by $\mathrm{Surp}(P)$, is equal to the sum of the surplus of all the operations on P: $\mathrm{Surp}(P) = d(P) - (2x_1(P) + 2x_3(P) + x_2(P))$. The path P is said to be *compressible* if $\mathrm{Surp}(P) \geq 0$.

To justify the formula given in the definition of $\mathrm{Surp}(P)$, note that $d(P)$ is the sum of the surplus of the category-1 and category-4 operations on P. Each side of a $(1, 3)$ branch has surplus -2, and the total surplus of the category-2 operations on P is $-2x_1(P) - 2x_3(P)$. The 2-side (resp. the 5-side) of a $(2, 5)$ branch has surplus -1 (resp. 0), and the total surplus of the category-3 operations on P is $-x_2(P)$. This justifies why the given formula for $\mathrm{Surp}(P)$ captures the total value of the surplus on the whole path P. Intuitively speaking, in comparison to a $(3, 5)$ branch, the 1-side (resp. the 3-side) of a $(1, 3)$ branch "loses" a value 2 in the parameter reduction when compared with the 3-side (resp. the 5-side) of the $(3, 5)$ branch, and the 2-side of a $(2, 5)$ branch "loses" a value 1 in the parameter reduction when compared with the 3-side of the $(3, 5)$ branch. On the other hand, the value $d(P)$ corresponds to the "extra" parameter reduction we gain in comparison to $(3, 5)$ branches. Therefore, the surplus $\mathrm{Surp}(P)$ of a path P measures how much the "extra" gain in the parameter value can make up for the losses along the path.

Proposition III.16 *Let $\mathcal{T}$ be the search tree for the algorithm* **VC3-solver** *on input (G, k). If every root-leaf path in $\mathcal{T}$ is compressible, then the number of leaves in $\mathcal{T}$ is bounded by r_0^k, where $r_0 \leq 1.194$ is the unique positive root of the polynomial $x^5 - x^2 - 1$.*

Proof. First note that according to the algorithm **VC3-solver**, each branch node in $\mathcal{T}$ is either a $(1, 3)$ branch, a $(2, 5)$ branch, or an (a, b) branch that is not worse than a $(3, 5)$ branch (see Lemma III.11). We say that a search tree $\mathcal{T}_0$ is *normalized* if: (1) for every 1-child node α in $\mathcal{T}_0$, the child of α is a leaf; and (2) every branch node in $\mathcal{T}_0$ is either a $(1,3)$, a $(2,5)$, or a $(3,5)$ branch. We can use the following procedure to convert a general search tree $\mathcal{T}$ into a normalized search tree $\mathcal{T}_0$, with a one-to-one correspondence between the root-leaf paths in the two trees, and such that the corresponding root-leaf paths in the two trees have the same surplus. Let the leaves of the original search tree $\mathcal{T}$ be $\alpha_1, \ldots, \alpha_t$. We first construct, based on the tree $\mathcal{T}$, a search tree $\mathcal{T}'$ with leaves $\alpha'_1, \ldots, \alpha'_t$, as follows. For each i, let the path from the root to the leaf α_i in $\mathcal{T}$ be P_i. If $d(P_i) = 0$, then leave the path P_i unchanged and let α'_i in $\mathcal{T}'$ be α_i. If $d(P_i) > 0$, then add to P_i a new leaf α'_i with label $d(P_i)$ and make α'_i the unique child of α_i (thus α_i becomes a 1-child non-leaf node in $\mathcal{T}'$). To obtain the normalized tree $\mathcal{T}_0$, we further perform the following two operations on $\mathcal{T}'$: (1) convert each (a,b) branch node α that is not worse than a $(3,5)$ branch into a $(3,5)$ branch by giving the label 3 (resp. the label 5) to the child of α corresponding to the a-side (resp. b-side) of α; (2) remove all non-branching nodes: for each 1-child node α in the tree with a child β, where β is not a new leaf created in $\mathcal{T}'$, remove the edge $[\alpha,\beta]$, merge the two nodes α and β, and assign a label to the resulting (new) node equal to the label of α (this corresponds to removing the non-branching operation specified by β). The resulting tree $\mathcal{T}_0$, with leaves $\alpha'_1, \ldots, \alpha'_t$, is a normalized search tree.

Let P_i be the path from the root to the leaf α_i in $\mathcal{T}$ and let P'_i be the path from the root to the leaf α'_i in $\mathcal{T}_0$. Since no $(1,3)$ branch nodes or $(2,5)$ branch nodes are changed or re-labeled in the above procedure, we have $x_1(P_i) = x_1(P'_i)$, $x_3(P_i) = x_3(P'_i)$, and $x_2(P_i) = x_2(P'_i)$. Moreover, if $d(P_i) = 0$, then the operations in steps (1) and (2) above are not applicable to P_i. Therefore, the path P'_i is the same as P_i, and $d(P'_i) = 0$. On the other hand, if $d(P_i) > 0$, then by our construction, the only node on P'_i that is not a $(1,3)$, a $(2,5)$, or a $(3,5)$ branch is the 1-child node whose child is the leaf α'_i with a label $d(P_i)$. Thus, $d(P'_i) = d(P_i)$, and the paths P_i

and P_i' have the same surplus.

From the above discussion, for each general search tree satisfying the condition in the proposition, there is a normalized search tree with the same number of leaves that also satisfies the condition in the proposition. Thus, it suffices to prove the proposition for normalized search trees. We do this by induction on the number of nodes in a normalized search tree $\mathcal{T}$. The proposition certainly holds true if the tree $\mathcal{T}$ consists of a single node or has only one leaf. Now assume that $|\mathcal{T}| > 1$ and that $\mathcal{T}$ has more than one leaf. Since $\mathcal{T}$ is normalized, the root α of $\mathcal{T}$ must be a branch node, which is either a $(1,3)$, a $(2,5)$, or a $(3,5)$ branch node.

Suppose the root α of $\mathcal{T}$ is a $(1,3)$ branch. Let β_1 and β_3 be the children of α labeled 1 and 3, respectively. Let $\mathcal{T}_1$ be the subtree rooted at β_1 in $\mathcal{T}$. Every path P_i from the root α to a leaf α_i in $\mathcal{T}_1$ contains the node β_1, and hence $x_1(P_i) \geq 1$. Since the path P_i is compressible, we have $\text{Surp}(P_i) = d(P_i) - (2x_1(P_i) + 2x_3(P_i) + x_2(P_i)) \geq 0$. It follows that $d(P_i) \geq 2$, and the label of the leaf α_i is at least 2. Therefore, in the tree $\mathcal{T}$ we can "shift" 2 units from the label of each leaf in the subtree $\mathcal{T}_1$ to the node β_1, by adding 2 units to the label of β_1 and subtracting 2 units from the label of every leaf in $\mathcal{T}_1$. Now the label of β_1 becomes 3. Similarly, we can add 2 units to the label of the node β_3 and subtract 2 units from the label of every leaf in the subtree rooted at β_3. This makes the label of β_3 become 5. Note that the resulting search tree is still normalized, with the difference that the root α now becomes a $(3,5)$ branch node, and that the label of each leaf in $\mathcal{T}$ is decreased by 2. In particular, each root-leaf path P_i in the resulting tree is still compressible (with the value $x_1(P_i)$ or $x_3(P_i)$ decreased by 1 and the value $d(P_i)$ decreased by 2).

Similarly, if the root α of the tree $\mathcal{T}$ is a $(2,5)$ branch with its label-2 child β_2 corresponding to the 2-side of the branch, then we can decrease the label of each leaf in the subtree rooted at β_2 by 1, add 1 to the label of β_2, and make the root α a $(3,5)$

branch. All root-leaf paths remain compressible.

Therefore, we can always end up with a normalized search tree $\mathcal{T}$ whose root is a $(3, 5)$ branch in which all root-leaf paths are compressible. Let γ_3 be the child of α labeled by 3 and γ_5 be the child of α labeled by 5. Consider the subtree $\mathcal{T}_3$ rooted at γ_3. By re-setting the label of γ_3 to 0, the subtree $\mathcal{T}_3$ becomes a valid normalized search tree for the algorithm **VC3-solver** on input $(G', k-3)$, where G' is the graph resulting from G by the operation specified by γ_3. Moreover, each root-leaf path in $\mathcal{T}_3$ is compressible since the node γ_3 in $\mathcal{T}$ is not a child of a $(1, 3)$ branch or a $(2, 5)$ branch node. Now by the inductive hypothesis, the number of leaves in $\mathcal{T}_3$ is bounded by r_0^{k-3}, where r_0 is the unique positive root of the polynomial $x^5 - x^2 - 1$. Similarly, re-setting the label of γ_5 to 0 makes the subtree rooted at γ_5 a valid normalized search tree with no more than r_0^{k-5} leaves. Adding the number of the leaves in the two subtrees, we get that the number of leaves in the search tree $\mathcal{T}$ is bounded by $r_0^{k-3} + r_0^{k-5}$. Since the polynomial $x^k - x^{k-3} - x^{k-5}$ and the polynomial $x^5 - x^2 - 1$ have the same positive root r_0, we get $r_0^k = r_0^{k-3} + r_0^{k-5}$, which proves that the number of leaves in the search tree $\mathcal{T}$ is bounded by r_0^k. This completes the inductive proof and the proof of the proposition. $\square$

By Proposition III.16, what remains to show is that every root-leaf path in a search tree for the algorithm **VC3-solver** is compressible. We start with the following proposition.

Proposition III.17 *Let $P = (\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+l})$, $l > 0$, be a subpath of a root-leaf path in a search tree $\mathcal{T}$ for the algorithm* **VC3-solver**. *If α_{i+l} is the only node on the path P whose associated graph is 3-regular, then the path P is compressible.*

PROOF. Let (G_{i-1}, k_{i-1}) be the instance associated with the parent node α_{i-1} of

α_i in $\mathcal{T}$, where the graph G_{i-1} has n_{i-1} vertices and m_{i-1} edges (recall that the root of $\mathcal{T}$ also has a virtual parent associated with the input instance to the algorithm). Let G_{i+l} be the graph associated with the node α_{i+l} where G_{i+l} has n_{i+l} vertices and m_{i+l} edges. Since the graph G_{i+l} is 3-regular, we have $m_{i+l}/n_{i+l} = 3/2$. Let $m' = m_{i-1} - m_{i+l}$, $n' = n_{i-1} - n_{i+l}$. Since $m_{i-1}/n_{i-1} \leq 3/2$ (the graph G_{i-1} has degree bounded by 3), we have $m'/n' \leq 3/2$.

Let x_f be the number of folding operations on P, E_f the number of edges removed, V_f the number of vertices removed, S_f the surplus, and K_f the reduction of the parameter, in all folding operations on P. In a similar way, define x_1, E_1, V_1, S_1, K_1, for the 1-side of the $(1,3)$ branches; x_3, E_3, V_3, S_3, K_3, for the 3-side of the $(1,3)$ branches; x_2, E_2, V_2, S_2, K_2 for the 2-side of the $(2,5)$ branches; x_5, E_5, V_5, S_5, K_5, for the 5-side of the $(2,5)$ branches; and x_r, E_r, V_r, S_r, K_r, for the category-4 operations on P. Since $m'/n' \leq 3/2$, we can write

$$\frac{E_f + E_1 + E_3 + E_2 + E_5 + E_r}{V_f + V_1 + V_3 + V_2 + V_5 + V_r} \leq \frac{3}{2}. \tag{3.8}$$

Arranging (3.8), we get

$$3V_f - 2E_f \geq (2E_1 - 3V_1) + (2E_3 - 3V_3) + (2E_2 - 3V_2) + (2E_5 - 3V_5) + (2E_r - 3V_r). \tag{3.9}$$

From the definition of the amortized cost, and by the monotonicity of addition, we can define the amortized cost for each type of operations, λ ($\lambda = 1, 2, 3, 5, r, f$), by: $M_\lambda = 5E_\lambda - 6V_\lambda + 6S_\lambda - 3K_\lambda$. Since the total K_λ vertices included in the partial cover for any type of operations λ must cover all the E_λ edges removed by that type, and since each vertex can cover at most three edges, $K_\lambda \geq E_\lambda/3$. Hence, $2E_\lambda - 3V_\lambda \geq -3S_\lambda + M_\lambda/2$. Using this inequality and the parameters of the operations given in Table I, we get: $3V_f - 2E_f \leq \frac{5}{2}x_f$, $2E_1 - 3V_1 \geq 3x_1$, $2E_3 - 3V_3 \geq 3x_3$, $2E_2 - 3V_2 \geq 3x_2$, $2E_5 - 3V_5 \geq \frac{1}{2}x_5$, $2E_r - 3V_r \geq -3S_r + M_r/2$. Substituting these bounds in Inequality (3.9) and arranging it we get:

$$x_f + S_r \geq x_2 + (x_1 + x_3) + x_5/6 + x_f/6 + M_r/6. \tag{3.10}$$

Since the graph G_{i+l} associated with the node α_{i+l} is 3-regular, it is not difficult to verify the following: Either we must have at least one folding operation along P, or at least one operation of those described in step 4 of **BindingSet2()**. This is true since these are the only operations that could make the graph become 3-regular. (The only way to create a 3-regular graph during the execution of the algorithm is either by a folding operation or by an operation in step 4 of **BindingSet2()**, which adds an edge to the resulting graph. All the other operations remove some vertices from the graph, which has degree bounded by 3, and hence cannot result in a 3-regular graph.) Since every category-4 operation has a non-negative amortized cost by Proposition III.13, and since the amortized cost of the operation in step 4 of **BindingSet2()** was proved to be at least 6 in Lemma III.9, it follows that if the operation in step 4 of **BindingSet2()** is performed, then we must have $M_r \geq 6$. Therefore, we either have $x_f \geq 1$, or $M_r \geq 6$. Since $x_f + S_r$ is an integer, from Inequality (3.10), we get

$$x_f + S_r \geq x_2 + (x_1 + x_3) + 1. \tag{3.11}$$

Note that if a node α specifies an operation corresponding to the 1-side or the 3-side of a $(1, 3)$ branch, then the graph associated with the parent of α must be 3-regular (see step 4 of the algorithm **VC3-solver**). Since α_{i+l} is the only node on the path P whose associated graph is 3-regular, and since α_{i+l} is the last node on the path, there is at most one node (i.e., node α_i) on the path P that may specify the 1-side or the 3-side operation of a $(1, 3)$ branch, and hence, $x_1 + x_3 \leq 1$. Combining this observation with Inequality (3.11), we get

$$x_f + S_r \geq x_2 + 2(x_1 + x_3). \tag{3.12}$$

Using the same notations given before Definition III.15, we have $d(P) = x_f + S_r$, $x_2 = x_2(P)$, $x_1 = x_1(P)$, $x_3 = x_3(P)$, and $(x_f + S_r) - (x_2 + 2(x_1 + x_3))$ is the surplus of the path P. Thus, Inequality (3.12) gives that $\text{Surp}(P) \geq 0$, and hence the path P is compressible. □

Proposition III.18 *Let G be a nice graph with n vertices and m edges. Then $m/n \geq 6/5$.*

Proof. The nice graph G contains no vertices of degree less than 2. Let n_2 and n_3 be the number of degree-2 and degree-3 vertices in G, respectively. Then $2m = 2n_2 + 3n_3 = 2n + n_3$. Since the nice graph G contains no adjacent degree-2 vertices, we have $3n_3 \geq 2n_2$. Combining these two relations we get the desired result. □

Lemma III.19 *Every root-leaf path in a search tree $\mathcal{T}$ of the algorithm* **VC3-solver** *is compressible.*

Proof. For an input (G, k) to the algorithm **VC3-solver**, if the graph G is 3-regular, then we subdivide an edge of G by two degree-2 vertices. Let the resulting graph be G'. Since the graph G can be obtained from G' by folding a degree-2 vertex in G', by Lemma IV.5, G has a vertex cover of size k if and only if G' has a vertex cover of size $k + 1$. Therefore, we can instead apply the algorithm to the instance $(G', k' = k + 1)$, where G' is not a 3-regular graph. Note that after subdividing an edge in G to obtain G', G' is connected, and $\tau(G') = \tau(G) + 1$. Since the parameter

k in the instance (G, k) is assumed to be not larger than $\tau(G)$ by condition (1) in Assumption III.5, the parameter k' is also not larger than $\tau(G')$. Therefore conditions (1) and (2) in Assumption III.5 still hold on the graph G'. Moreover, since G' has two more vertices than G, and the parameter $k' = k + 1$, G' satisfies the assumption given by Proposition II.1, namely that the number of vertices in G' is bounded by $2k'$. By doing this operation, the order of the running time of the algorithm is not affected. Thus, we can always assume that the graph associated with the root of the search tree $\mathcal{T}$ is not a 3-regular graph.

For any root-leaf path $P' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_t)$ in the search tree $\mathcal{T}$, let α'_{i_1}, α'_{i_2}, $\ldots$, α'_{i_r} be the nodes on P' whose associated graphs are 3-regular. Note that it is impossible for two graphs associated with two consecutive nodes on P' to be both 3-regular — the only operation applicable to a 3-regular graph is step 4 in the algorithm **VC3-solver** that does not result in a 3-regular graph. Thus, each of the subpaths $(\alpha'_{i_{j-1}+1}, \ldots, \alpha'_{i_j})$ on P', $j = 1, \ldots, r$ (here we let α'_{i_0+1} be α'_1), satisfies the condition in Proposition III.17 and is compressible, which equivalently means that the path has a non-negative surplus. Therefore, in order to prove the lemma, it suffices to show that the subpath $P = (\alpha'_{i_r+1}, \ldots, \alpha'_t)$ has a non-negative surplus, and hence is compressible. To simplify the notations, we rename the nodes on P and let $P = (\alpha_1, \alpha_2, \ldots, \alpha_s)$ (if the root-leaf path P' contains no node associated with a 3-regular graph, we let $P = P'$).

Let (G_0, k_0) be the instance associated with the parent of α_1 (note that the root of $\mathcal{T}$ also has a virtual parent whose associated instance is the original input to the algorithm), where the graph G_0 has n_0 vertices and m_0 edges. Since the degree of G_0 is bounded by 3, we have

$$m_0/n_0 \leq 3/2. \tag{3.13}$$

If α_1 is the root of $\mathcal{T}$, then (G_0, k_0) is the original input instance to the algorithm.

In this case, by Proposition II.1, we have $k_0 \geq n_0/2$. On the other hand, If α_1 is not the root of $\mathcal{T}$, then the graph G_0 associated with the parent node of α_1 is 3-regular, and $2m_0 = 3n_0$. Since each vertex can cover at most 3 edges, we must have $3k_0 \geq m_0$ (otherwise the answer to the instance (G_0, k_0) is negative). This also gives us $k_0 \geq n_0/2$. Therefore, in all cases, we have

$$k_0 \geq n_0/2. \tag{3.14}$$

Case 1. All the nodes on the path P are non-branching nodes.

Suppose that the parameter reduction and the surplus of the operation specified by the first node α_1 on P are k_1 and s_1, respectively. By the definition of the surplus, we have $s_1 \geq k_1 - 5$. The instance associated with α_1 is $(G_1, k_0 - k_1)$ for some graph G_1. By condition (1) in Assumption III.5, the original parameter k is not larger than the size of a minimum vertex cover of G. Therefore the parameter value $k_0 - k_1$ associated with α_1 is not larger than the size of a minimum vertex cover of G_1, otherwise the original parameter k would be larger than the size of a minimum vertex cover of G. It follows that when the algorithm terminates at node α_s along the path P in the search tree, either the computed cover is a minimum vertex cover, and hence, the reduction in the parameter along the path $\alpha_2, \ldots, \alpha_s$ is exactly equal to $k_0 - k_1$; or the size of the resulting cover for G_1 has exceeded the parameter $k_0 - k_1$, and hence, the reduction in the parameter along the path $\alpha_2, \ldots, \alpha_s$ is greater than $k_0 - k_1$. Note that all the nodes on P, except α_1, specify non-branching operations whose parameter reduction and surplus are equal. Therefore, the reduction in the parameter, or equivalently the sum of the surplus, of the nodes $\alpha_2, \ldots, \alpha_s$ is at least $k_0 - k_1$. Adding the surplus of the node α_1, we get

$$\text{Surp}(P) \geq s_1 + (k_0 - k_1) \geq (k_1 - 5) + (k_0 - k_1) = k_0 - 5. \tag{3.15}$$

Observe that we have $k_0 \geq 25$. In fact, if $k_0 \leq 24$, then by Inequality (3.14), $n_0 \leq 2k_0 < 50$. In such case step B of **Reducing** would be applicable to (G_0, k_0), and the parent node of α_1 would not be a branch node. Combining the fact that $k_0 \geq 25$ with Inequality (3.15), we conclude that in this case $\mathrm{Surp}(P) \geq 20$.

Case 2. The path P contains branch nodes. Let α_h be the last branch node on P.

Consider the two subpaths $P_1 = (\alpha_1, \ldots, \alpha_h)$ and $P_2 = (\alpha_{h+1}, \ldots, \alpha_s)$ of P. Let (G_h, k_h) be the instance associated with the node α_h. Since all nodes in the subpath P_2 are non-branching nodes, as shown in Case 1 (see Inequality (3.15)), we have

$$\mathrm{Surp}(P_2) \geq k_h - 5. \tag{3.16}$$

Now consider the value $\mathrm{Surp}(P_1)$. Let x_f, E_f, V_f, K_f, S_f, x_1, E_1, V_1, K_1, S_1, x_3, E_3, V_3, K_3, S_3, x_2, E_2, V_2, K_2, S_2, x_5, E_5, V_5, K_5, S_5, x_r, E_r, V_r, K_r, S_r, denote the same entities as in Proposition III.17 along the subpath $P_1 = (\alpha_1, \ldots, \alpha_h)$. We have

$$x_f + x_1 + 2x_2 + 3x_3 + 5x_5 + K_r + k_h = k_0. \tag{3.17}$$

This is because the operation specified by the first node α_1 on the subpath P_1 is applied to the instance (G_0, k_0) (which is associated with the parent of α_1), and the last node α_h on P_1 is associated with the instance (G_h, k_h), and $x_f + x_1 + 2x_2 + 3x_3 + 5x_5 + K_r$ is the total parameter reduction of the operations on P_1.

According to our algorithm, the graph G_h associated with the branch node α_h in the search tree $\mathcal{T}$ is nice. Thus, if we let n_h and m_h be the number of vertices and edges in G_h, respectively, then by Proposition III.18, $m_h/n_h \geq 6/5$. Using our notations, we have $m_h = m_0 - E_f - E_1 - E_3 - E_2 - E_5 - E_r$ and $n_h = n_0 - V_f - V_1 - V_3 - V_2 - V_5 - V_r$. Therefore,

$$\frac{m_0 - E_f - E_1 - E_3 - E_2 - E_5 - E_r}{n_0 - V_f - V_1 - V_3 - V_2 - V_5 - V_r} \geq \frac{6}{5}. \tag{3.18}$$

In a similar way to that in Proposition III.17, define the amortized cost for each type of operations λ ($\lambda = 1, 2, 3, 5, r, f$), by: $M_\lambda = 5E_\lambda - 6V_\lambda + 6S_\lambda - 3K_\lambda$. Hence, we have $5E_\lambda - 6V_\lambda = M_\lambda + 3K_\lambda - 6S_\lambda$. Using this equality and the parameters of the operations given in Table I, we get: $6V_f - 5E_f \leq 2x_f$, $5E_1 - 6V_1 \geq 9x_1$, $5E_3 - 6V_3 \geq 15x_3$, $5E_2 - 6V_2 \geq 12x_2$, $5E_5 - 6V_5 \geq 16x_5$, $5E_r - 6V_r \geq M_r + 3K_r - 6S_r$. Combining these inequalities with Inequalities (3.13), (3.14), (3.17), (3.18), and arranging the terms we get:

$$5x_f \geq 6x_1 + 6x_2 + 6x_3 + x_5 + M_r - 6S_r - 3k_h. \tag{3.19}$$

Hence:

$$x_f + S_r \geq x_2 + (x_1 + x_3) + x_5/6 + M_r/6 + x_f/6 - k_h/2 \geq x_2 + (x_1 + x_3) - k_h/2. \tag{3.20}$$

Here we have used Proposition III.13 which gives that $M_r \geq 0$. Since $d(P_1) = x_f + S_r$, $x_1 = x_1(P_1)$, $x_2 = x_2(P_1)$, and $x_3 = x_3(P_1)$ (see Definition III.15), From (4.6), we get

$$\text{Surp}(P_1) = (x_f + S_r) - (x_2 + 2(x_1 + x_3)) \geq -(x_1 + x_3) - k_h/2. \tag{3.21}$$

Combining this inequality with Inequality (3.16),

$$\text{Surp}(P) = \text{Surp}(P_1) + \text{Surp}(P_2) \geq k_h/2 - (x_1 + x_3) - 5. \tag{3.22}$$

As explained in Case 1, since the node α_h is a branch node, the graph G_h associated with α_h must be nice and have at least 50 vertices. Since G_h is nice (and hence is clean), the number of edges in G_h is more than 50. Since each vertex can cover at most 3 edges, we have $k_h \geq 17$ (otherwise the answer to the instance (G_h, k_h) is negative). Moreover, no graph associated with a node on the path P_1 is 3-regular. Since a 1-side or a 3-side operation of a $(1, 3)$ branch can only be applied on a 3-regular graph, there is at most one node on P_1 (the node α_1) that may specify such an operation. Therefore, $x_1 + x_3 \leq 1$. Combining the last two inequalities with Inequality (3.22), we get $\text{Surp}(P) \geq 0$ and the path P is compressible. This completes the proof of the lemma. $\square$

Theorem III.20 *The algorithm* **VC3-solver** *runs in time* $O(1.194^k + n)$.

Proof. First observe that by spending $O(n)$ time in the subroutine **Clean**, we can remove vertices of degree 0 and 1. After that, it must be true that every component in the graph is a non-tree component, and hence, at least one third of the number of vertices in each component must be included in any vertex cover of the component. This means that the resulting parameter k satisfies $k \geq n/3$, where n is the number of vertices in the resulting graph (otherwise the answer to the instance is negative). Then the algorithm mentioned in Proposition II.1 is applied. This algorithm runs in $O(k\sqrt{k})$ time. Finally the algorithm **VC3-solver** is invoked.

Let $\mathcal{T}$ be the search tree for the algorithm **VC3-solver** on the input instance. By Lemma III.19, every root-leaf path in $\mathcal{T}$ is compressible. Since every branching operation in $\mathcal{T}$ can be classified as a $(1,3)$, $(2,5)$, or (a,b), with (a,b) not worse than a $(3,5)$-branch, from Proposition III.16 we get that the number of leaves in $\mathcal{T}$ is $O(r^k)$, where $r \leq 1.194$ is the positive root of the polynomial $x^5 - x^2 - 1$. It follows that the number of root-leaf paths in $\mathcal{T}$ is also $O(1.194^k)$.

At every node in the search tree $\mathcal{T}$ the time spent by the algorithm is linear in the size of the graph, which is $O(k)$. To verify that, let us look at the operations performed by the algorithm. First, whenever **Clean** is invoked, the time spent is proportional to the size of the subgraph removed, and hence is $O(k)$. Also the time taken by a branching operation is $O(k)$. We explained in Section B how step 2 of **VC3-solver** can be carried out in time $O(k)$. We also explained in Section B how each step in **Reducing** can be performed in time $O(k)$. This shows that the time spent at every node in the search tree is $O(k)$.

By the standard analysis using the *interleaving technique* introduced by Niedermeier and Rossmanith [43], the running time of the algorithm is bounded by $O(1.194^k + k^2 + n) = O(1.194^k + n)$, where $O(k\sqrt{k} + n)$ is the pre-processing time. This completes the proof. $\square$

D. An Algorithm for IS-3

In this section we show how the algorithm for VC-3 implies an algorithm for IS-3. The approach is exactly the same as that employed in [12], which used a less efficient algorithm for VC-3 than the one given in this chapter, to derive an algorithm for IS-3 running in time $O(1.174^n)$. The algorithm for IS-3 presented here runs in time $O(1.1255^n)$, and slightly beats the previously most efficient $O(1.1259^n)$-time algorithm by Beigel [45].

Lemma III.21 (Lemma 6.1, [12]) *Let G be a connected graph of n vertices and degree bounded by 3. Then a minimum vertex cover of G contains at most $(2n+1)/3$ vertices.*

Theorem III.22 *The IS-3 problem can be solved in time $O(1.1255^n)$.*

Proof. Let G be a graph of degree bounded by 3. The graph G may not necessarily be connected. Let C_1, ..., C_k be the connected components of G of sizes n_1, ..., n_k, respectively. It is clear that a maximum independent set of G is the union of maximum independent sets of the components $C_1, \cdots, C_k$. For each component C_i of G, instead of finding a maximum independent set for C_i, we try to construct a vertex cover of k_i vertices, for $k_i = 1, 2, \ldots$. At the first k_i for which we are able to construct a vertex cover of k_i vertices for C_i, we know this vertex cover is a minimum vertex cover. Thus, the complement of this vertex cover is a maximum independent set for C_i. By

Lemma III.21, we must have $k_i \leq (2n_i + 1)/3$. Thus, by Theorem III.20, a maximum independent set for the component C_i can be constructed in time $O(1.194^{(2n_i+1)/3} + n_i)$, which is $O(1.1255^{n_i})$. In conclusion, a maximum independent set in the graph G can be constructed in time $O(1.1255^{n_1} + \cdots + 1.1255^{n_k})$. By an argument similar to that given in the proof of Assumption III.5 at the end of Section B, it follows that $O(1.1255^{n_1} + \cdots + 1.1255^{n_k}) = O(1.1255^{n})$. □

E. Comments

In this chapter we presented algorithms for the parameterized Vertex Cover and the Maximum Independent Set problems on degree-3 graphs. Our algorithm for VC-3 runs in time $O(1.194^k k^2 + n)$ and improves Chen et al.'s $O(1.237^k + kn)$ time algorithm [23]. Our algorithm for IS-3 runs in time $O(1.1255^n)$ and improves Beigel's $O(1.1259^n)$ time algorithm [45].

We emphasize that the importance of our results lies in the techniques that we use to analyze the size of the search tree. Despite the fact that the analysis of the algorithm is lengthy, the algorithm itself is very simple and uniform. The algorithm distinguishes few cases to eliminate cut-vertices and bridges from the graph. However, all these cases are solved easily and without any branching. As a matter of fact, these cases use very simple and elegant graph-theoretic operations that can be generalized in a straightforward manner to the Vertex Cover problem on general graphs. If one looks carefully at the algorithm itself, the algorithm is very intuitive. Basically the overall behavior of the algorithm can be described as follows. As long as the case can be solved without any branching, solve it (folding, reducing, and cleaning). If none of the above applies, then either we can do an efficient and uniform branch (alternating tree), which is a single branch that does not distinguish any cases, or we branch arbitrarily at any vertex, and the amortized analysis shows that this operation will be balanced by non-branching operations. The analysis of the algorithm might be lengthy, but the techniques involved are elementary combinatorial techniques.

CHAPTER IV

PROBLEMS ON GRAPHS WITH CONSTRAINED GENUS

In this chapter, we demonstrate how the genus of the underlying graph plays an important role in characterizing the parameterized complexity, the subexponential time computability, and the approximability of the Vertex Cover, Independent Set, and Dominating Set problems.

A. Computational Complexity and Graph Genus

Variants of these problems were studied where the input graph is constrained to have certain structural properties (e.g., bounded degree graphs and planar graphs) [52, 32, 2, 33, 6]. In particular, the problems on the class of planar graphs (the problems remain NP-hard) become more tractable in terms of the above three complexity measures. All of the three problems on planar graphs have polynomial time approximation schemes [53, 54], and are solvable in subexponential time [54]. Recent research in fixed parameter tractability shows that all the three problems admit parameterized algorithms whose running time is subexponential in the parameter [32]. This line of research has attracted considerable recent interests and the results have been extended to graphs of bounded genus [55, 33, 56].

This raises an interesting question: What are the graph structures that determine the computational complexity of these important NP-hard problems?

Our research shows that in most cases, graph genus is the sole factor that determines the complexity of the above problems. More precisely, in most cases, there is a precise genus threshold that determines the computational complexity of the problems in terms of the three complexity measures. For instance, we show that under the widely-believed complexity assumption $W[2] \neq$ FPT, Dominating Set is fixed parameter tractable *if and only if* the graph genus is $n^{o(1)}$. This result significantly extends both Alber *et al.* and Ellis *et al.*'s results for planar graphs and for constant genus graphs [52, 33]. The proof is also simpler and more uniform. It is also shown that under the assumption $W[1] \neq \text{FPT}$, Independent Set is fixed parameter tractable *if and only if* the graph genus is $o(n^2)$. For the subexponential time computability, we show that under the assumption that not all SNP problems are solvable in subexponential time, Vertex Cover, Independent Set, and Dominating Set are solvable in subexponential time *if and only if* the genus of the graph is $o(n)$. In terms of approximability, we show that graph genus has a direct impact on whether Independent Set, Vertex Cover, and Dominating Set have polynomial time approximation schemes. A summary of our main results and the previous known results is given in Table II.

We make two remarks on our results. First, all our tractability results are robust [57] in the sense that our algorithms work correctly regardless of whether the input graphs satisfy the required genus bound $g(n)$. As long as the input graphs satisfy the required genus bound $g(n)$, our algorithms construct correct solutions for the problems; whereas when our algorithms fail in constructing a solution, they correctly report that the genus of the input graph exceeds the required bound $g(n)$. Second, the techniques proposed in the current chapter are not restricted to only the above three problems, and can be extended to derive similar results for other NP-hard graph problems.

We give a quick review on the related terminologies. A *surface* of genus g is a sphere with g handles in the 3-space [58]. A graph G *embedded* in a surface S is a continuous one-to-one mapping from the graph into the surface. The embedding is *cellular* if each component of $S - G$, which is called a *face*, is homeomorphic to an open disk [58]. In this chapter, we only consider cellular graph embeddings. The *size* of a face is the number of edge sides along the boundary of the face. The (*minimum*)

Table II. Comparison between our results and the previous results

	FPT		**Subexp. Time**		**Approximability**	
	Our Results	Previous Results	Our Results	Previous Results	Our Results	Previous Results
VC	–	FPT [11]	$2^{o(n)}$ iff $g{=}o(n)$	$2^{O(\sqrt{n})}$ if $g{=}c$ [32, 54]	PTAS¶ if $g{=}o(\frac{n}{\log n})$ APX-C if $g{=}n^{\Omega(1)}$	PTAS if $g{=}c$ [53, 54]
IS	FPT iff $g{=}o(n^2)$	FPT if $g{=}0$ [32]	$2^{o(n)}$ iff $g{=}o(n)$	$2^{O(\sqrt{n})}$ if $g{=}c$ [32, 54]	PTAS if $g{=}o(\frac{n}{\log n})$ APX-H if $g{=}\Omega(n)$	PTAS if $g{=}c$ [53, 54]
DS	FPT iff $g{=}n^{o(1)}$	FPT if $g{=}c$ [33]	$2^{o(n)}$ iff $g{=}o(n)$	$2^{O(\sqrt{n})}$ if $g{=}c$ [32, 54]	PTAS¶ if $g{=}o(\frac{n}{\log n})$ APX-H if $g{=}n^{\Omega(1)}$	PTAS if $g{=}c$ [53, 54]

¶Only true for kernelized graphs, see Theorem IV.17 and Theorem IV.18.

genus $\gamma_{\min}(G)$ of a graph G is the smallest integer g such that G has an embedding on a surface of genus g. For more detailed discussions on data structures and algorithms for graph embedding on surfaces, the readers are referred to [59].

B. Genus and Parameterized Complexity

1. Genus and Independent Set

The parameterized Independent Set problem (or simply Independent Set without any confusion) is a representative of the $W[1]$-complete problems [11]. Thus, it is unlikely to be fixed parameter tractable. Actually, very recent research has shown strong evidence that it is even unlikely that the problem is solvable in time $n^{o(k)}$ [60, 61]. In this subsection, we discuss how graph genus affects the parameterized complexity of Independent Set.

Theorem IV.1 *The Independent Set problem on graphs of genus bounded by $g(n)$ is fixed parameter tractable if $g(n) = o(n^2)$.*

Proof. Since $g(n) = o(n^2)$, there is a nondecreasing and unbounded function

$r(n)$ such that $g(n) \leq n^2/r(n)$.[1] Without loss of generality, we can assume that $r(n) \leq n^2$. Otherwise, $g(n) = 0$, and the theorem follows from [32]. Let G be a graph of n vertices and genus $g' \leq g(n)$. Recall that the *chromatic number* $\chi(G)$ of G is the smallest integer p such that G can be colored with p colors so that no two adjacent vertices are colored with the same color. By Heawood's Theorem [58], the chromatic number $\chi(G)$ of the graph G is bounded by $(7 + \sqrt{1 + 48g'})/2$. From the definition, the chromatic number $\chi(G)$ of G implies an independent set of at least $n/\chi(G)$ vertices in G. Thus, the size $\alpha(G)$ of a maximum independent set in the graph G is at least $2n/(7 + \sqrt{1 + 48g'})$. Since $g' \leq g(n) \leq n^2/r(n)$, we get (note that $r(n) \leq n^2$)

$$\alpha(G) \geq \frac{2n}{7 + \sqrt{1 + 48n^2/r(n)}} = \frac{2n\sqrt{r(n)}}{7\sqrt{r(n)} + \sqrt{r(n) + 48n^2}} \geq \frac{2n\sqrt{r(n)}}{7n + \sqrt{n^2 + 48n^2}} = \frac{\sqrt{r(n)}}{7} \tag{4.1}$$

Now we are ready for describing our parameterized algorithm. Note that one difficulty we must overcome is estimating the genus of the input graph. The graph minimum genus problem is NP-complete [62], and there is no known effective approximation algorithm for the problem. Therefore, some special tricks have to be used for this purpose. Here we will make use of the approximation algorithm for the graph minimum genus problem proposed in [63], which on an input graph G constructs an embedding of G whose genus is bounded by $\max\{4\gamma_{\min}(G), \gamma_{\min}(G) + 4n\}$. Consider the algorithm given in Figure 8.

We analyze the time complexity of the algorithm **IS-FPT**. First note that by our assumption on the function $r(n)$, the function $r_1(n)$ is also nondecreasing and unbounded. The embedding $\pi(G)$ of the graph G in step 2 can be constructed in

[1]In this chapter, we only consider "simple" complexity functions whose value can be feasibly computed. Thus, in our discussion, the computational time for computing the values of complexity functions as such $g(n)$ and $r(n)$ will be neglected.

```
ALGORITHM. IS-FPT
Input: a graph $G$ of $n$ vertices and an integer $k$
Output: decide if $G$ has an independent set of $k$ vertices
1. let $r_1(n) = \min\{r(n)/4, nr(n)/(n + 4r(n))\}$;
2. construct an embedding $\pi(G)$ of $G$ using the algorithm in [63];
3. if the genus of $\pi(G)$ is larger than $n^2/r_1(n)$ then Stop ("the genus of $G$ is larger than
   $g(n)$");
4. if $k \le \sqrt{r_1(n)}/7$ then Stop ("the graph $G$ has an independent set of $k$ vertices")
                     else try all vertex subsets of $k$ vertices to derive a conclusion.
```

Fig. 8. A parameterized algorithm for Independent Set

linear time [63], and the genus of the embedding $\pi(G)$ can also be computed in linear time [59].

Since $r_1(n) = \min\{r(n)/4, nr(n)/(n + 4r(n))\}$, if the genus $\gamma(\pi(G))$ of the embedding $\pi(G)$ is larger than $n^2/r_1(n)$, then $\gamma(\pi(G))$ is larger than both $4n^2/r(n)$ and $n^2/r(n) + 4n$. According to [63], the genus $\gamma(\pi(G))$ of the embedding $\pi(G)$ is bounded by $\max\{4\gamma_{\min}(G), \gamma_{\min}(G)+4n\}$. Thus, in case $\gamma(\pi(G)) \le 4\gamma_{\min}(G)$, we have $4\gamma_{\min}(G) > 4n^2/r(n)$, and in case $\gamma(\pi(G)) \le \gamma_{\min}(G) + 4n$, we have $\gamma_{\min}(G) + 4n > n^2/r(n) + 4n$. Thus, in all cases, we will have $\gamma_{\min}(G) > n^2/r(n) \ge g(n)$. In consequence, the algorithm **IS-FPT** concludes correctly if it stops in step 3.

If the algorithm **IS-FPT** reaches step 4, we know that the minimum genus of the graph G is bounded by $n^2/r_1(n)$. By the above analysis and the relation in (4.1), the size of a maximum independent set in G is at least $\sqrt{r_1(n)}/7$. Thus, in case $k \le \sqrt{r_1(n)}/7$, there must be an independent set in G with k vertices. On the other hand, if $k > \sqrt{r_1(n)}/7$ then $\overline{r_1}(49k^2) \ge n$, where $\overline{r_1}$ is the inverse function of the function $r_1(n)$ defined by $\overline{r_1}(p) = \min\{\ q \mid r_1(q) \ge p\ \}$. Since the function $r_1(n)$ is nondecreasing and unbounded, it is not difficult to see that the inverse function $\overline{r_1}(p)$ is also nondecreasing and unbounded. Since enumerating all vertex subsets of k vertices in the graph G can be done in $O(2^n)$ time, which is bounded by $O(2^{\overline{r_1}(49k^2)})$, we conclude that the total running time of the algorithm **IS-FPT** is bounded by $O(f(k)+n^2)$, where $f(k) = 2^{\overline{r_1}(49k^2)}$ is a function dependent only on k and not on n.

Thus, the algorithm **IS-FPT** solves the Independent Set problem on graphs of genus bounded by $g(n)$ in time $O(f(k)+n^2)$, and the problem is fixed parameter tractable. □

Remark. The algorithm **IS-FPT** does not have to know whether the input graph has its minimum genus bounded by $g(n)$. Moreover, the algorithm **IS-FPT** does not need to decide precisely whether the input graph has a minimum genus bounded by $g(n)$. In fact, on some graphs whose minimum genus is larger than $g(n)$, the algorithm **IS-FPT** may still be able to decide correctly whether the graphs have an independent set of size k. The point is, if the input graph has its minimum genus bounded by $g(n)$, then the algorithm **IS-FPT**, without needing to know this fact, will definitely and correctly decide whether it has an independent set of size k.

Theorem IV.2 *The Independent Set problem on graphs of genus bounded by $g(n)$ is $W[1]$-complete if $g(n) = \Omega(n^2)$.*

Proof. Since Independent Set on general graphs is $W[1]$-complete [11], it suffices to show that Independent Set on general graphs is fpt-reducible to Independent Set on graphs of genus bounded by $g(n)$. Since $g(n) = \Omega(n^2)$, we assume $g(n) \geq cn^2$, where c is a constant.

Let G_1 be an arbitrary graph with n_1 vertices. It is well-known that the genus g_1 of G_1 is always bounded by $(n_1-3)(n_1-4)/12 \leq n_1^2/12$ [58]. Thus, if $c \geq 1/12$ then G_1 already has its genus bounded by cn_1^2. Otherwise, we construct a new graph G_2 as follows. G_2 contains $h = \lceil 1/(12c) \rceil > 1$ copies of the graph G_1. Partition the h copies of G_1 arbitrarily into two nonempty groups $\mathcal{A}_1$ and $\mathcal{A}_2$, and pick any pair of adjacent vertices u_1 and v_1 in G_1. Now introduce a new edge $[u_2, v_2]$, where u_2 and v_2 are two new vertices. Connect u_2 to the vertex u_1 in each copy of G_1 in the group $\mathcal{A}_1$ and connect v_2 to the vertex v_1 in each copy of G_1 in the group $\mathcal{A}_2$. This completes the construction of the graph G_2. It is not difficult to verify that the graph G_1 has an independent set of k_1 vertices if and only if the graph G_2 has an independent set of $k_2 = hk+1$ vertices. Thus, the reduction from (G_1, k_1) to (G_2, k_2) is an fpt-reduction. Moreover, the graph G_2 has $n_2 = hn_1 + 2$ vertices and we can verify [58] that the genus of G_2 is $g_2 = hg_1$. Thus, we have

$$g_2 = hg_1 \leq \frac{hn_1^2}{12} = \frac{(hn_1)^2}{12h} \leq \frac{n_2^2}{12/(12c)} = cn_2^2 \leq g(n_2)$$

Thus, the genus of the graph G_2 of n_2 vertices is bounded by $g(n_2)$.

This completes the fpt-reduction that reduces an instance (G_1, k_1) of Independent Set on general graphs to an instance (G_2, k_2) of Independent Set on graphs of genus bounded by $g(n)$. In consequence, Independent Set on graphs of genus bounded by $g(n)$ is $W[1]$-complete. □

Combining Theorem IV.1 and Theorem IV.2, and noting that the genus of a graph of n vertices is always bounded by $(n-3)(n-4)/12$ [58], we have the following tight result.

Corollary IV.3 *Assuming $FPT \neq W[1]$, the Independent Set problem on graphs of genus bounded by $g(n)$ is not fixed parameter tractable if and only if $g(n) = \Theta(n^2)$.*

2. Genus and Dominating Set

Dominating Set is the most well-known $W[2]$-complete problem [11]. Thus, it is even "harder" than Independent Set in terms of its parameterized complexity. Recently, there has been considerable interest in developing parameterized algorithms for Dominating Set on graphs of small genus [52, 32, 55, 64, 33, 65, 56, 66]. In particular, it is known that Dominating Set on planar graphs [52, 32] and on graphs of constant genus [55, 64, 33, 56] is fixed parameter tractable. We will show a much stronger result in this subsection: Dominating Set on graphs of genus bounded by $g(n)$ is fixed-parameter tractable if and only if $g(n) = n^{o(1)}$.

For a given instance (G, k) of Dominating Set, we apply a branch-and-bound process to construct a dominating set D of k vertices in G. Initially, $D = \emptyset$. In a more general form during the process, suppose we have correctly included certain vertices in the dominating set D, and removed these vertices from the graph G. The vertices in the remaining graph G' are colored either "white" or "black", where each white vertex is adjacent to a vertex in D (thus needs no further domination) and each black vertex is adjacent to no vertex in D (thus still needs to be dominated in the remaining graph G'). The graph G' thus will be called a *BW-graph*. We call a set D' of vertices in the BW-graph G' a *B-dominating set* if every black vertex in G' is either in D' or is adjacent to a vertex in D'. Note that if the current set D has d vertices, then the graph G has a dominating set of k vertices, including all vertices in D, if and only if the BW-graph G' has a B-dominating set of $k - d$ vertices. Thus, our task is to construct a B-dominating set of $k - d$ vertices in the BW-graph G'.

Certain reduction rules can be applied to a BW-graph G':

R1. Remove from G' all edges between white vertices;

R2. Remove from G' all white vertices of degree 1;

R3. If all neighbors of a white vertex u_1 are neighbors of another white vertex u_2, remove u_1 from G'.

Let G'' be a BW-graph after applying any of the above rules on G'. It is known [52, 33] that there is a B-dominating set of k vertices in G' if and only if there is a B-dominating set of k vertices in G''. A BW-graph G is called *reduced* if none of the above rules can be applied. According to rule **R1**, every edge in a reduced BW-graph either connects two black vertices or connects a black vertex and a white vertex (the edge will be called a *bb-edge* or a *bw-edge*, respectively).

We will show that in a reduced BW-graph, the number of black vertices will not be very small. For this purpose, we first need to give a brief discussion on certain basic facts about graph embeddings. For more detailed and formal proofs of these facts, the readers are referred to [59].

Fact 1. A face of size 1 can only be made by a self-loop, and a face of size 2 must be made by two multiple edges on the same pair of vertices.

Fact 2. Let F be a face of size d in a graph embedding with boundary vertices u_1, u_2, …, u_d, cyclically ordered along the face boundary. If we run a new edge from u_1 to u_i crossing the face F, $1 \leq i \leq d$, then the face F is split into two faces of sizes i and $d-i+2$, respectively, both having the new edge on their face boundaries. No other faces in the embedding are changed. Moreover, the embedding genus is unchanged.

Fact 3. In a given embedding of a graph G, the neighbors of every vertex u in G specify a unique cyclic order $[u_1, u_2, \ldots, u_d]$ so that the edges $[u, u_1]$, $[u, u_2]$, …, $[u, u_d]$ form a cyclic order around the vertex u in a small region on the embedding. In particular, if every triple (u, u_i, u_{i+1}), $i = 1, 2, \ldots, d$ (here we take u_{d+1} as u_1), makes a triangle face on the embedding, then removing the vertex u (and all edges incident on u) will merge all these triangle faces into a single face of size d. The embedding genus and all other faces are unchanged.

Fact 4. Suppose there is a triangle face (u_1, u_2, u_3) in an embedding, the vertex u_1 has degree 2, and there are no multiple edges between u_2 and u_3, then removing the vertex u_1 and the two edges incident on u_1 neither changes the embedding genus nor creates a face of size less than 3.

The following lemma can be easily derived from the famous Euler Polyhedral Equation [58].

Lemma IV.4 *If G is a graph of n vertices and m edges (with possibly multiple edges and self-loops), and G has an embedding on a surface of genus g such that all faces of the embedding have size at least 3, then $m \leq 6g + 3n - 6$.*

Now we are ready to prove the following important lemma, which derives relations among the numbers of black vertices, white vertices, edges, and the genus of a reduced BW-graph.

Lemma IV.5 *Let G be a reduced BW-graph of minimum genus g, with m edges and n vertices, in which n_w are white and n_b are black, and suppose that G has neither multiple edges nor self-loops, then* (a) $m \leq 9n_b + 18g - 18$*; and* (b) $n \leq 4n_b + 6g - 6$.

Proof. Let $\pi(G)$ be an embedding of genus g for the graph G. By rules **R1** and **R2**, the degree of a white vertex u in G is at least 2 and all neighbors of u are black. We perform the following operations on each white vertex u.

If the white vertex u has degree 2 with two black neighbors u_1 and u_2, and there is no edge between u_1 and u_2, then we add a new edge $[u_1, u_2]$ crossing a face in the embedding to make a triangle face with u (note that since u has degree 2, this is always possible). Adding the new edge $[u_1, u_2]$ does not create a face of size less than 3, because it does not introduce new self-loops or new multiple edges. Moreover, the embedding genus is unchanged.

If u has degree $d > 2$ and u_1, u_2, …, u_d are the d black neighbors of u, ordered in clockwise order around u in the embedding, then for each pair of vertices u_i and u_{i+1}, $i = 1, 2, \ldots, d$ (here we take u_{d+1} as u_1), if the vertices u, u_i, u_{i+1} do not form a triangle face in the embedding $\pi(G)$, then we add a new edge $[u_i, u_{i+1}]$, crossing a face in the embedding $\pi(G)$, to make a triangle face (u, u_i, u_{i+1}) (again, this is always possible). This does not change the embedding genus. Note that adding this new edge may create multiple edges between u_i and u_{i+1}. However, the new edge does not create any faces of size less than 3. This can be proved as follows. First this does not create faces of size 1 because it does not create self-loops. Second, if it created a face of size 2, then the two sides of the new edge $[u_i, u_{i+1}]$ are on the face boundaries of a face of size 2 and a face of size 3 (i.e., the triangle face (u, u_i, u_{i+1})). This, according to Fact 2, would imply that before adding the new edge $[u_i, u_{i+1}]$, the vertices u, u_i, u_{i+1} had already made a triangle face. This proves that adding the new edge $[u_i, u_{i+1}]$ does not create faces of size less than 3. Finally, note that the vertices u_i and u_{i+1} cannot be the neighbors of a white vertex of degree 2 – otherwise by rule **R3**, the white vertex of degree 2 would have been removed. Thus, processing white vertices of degree larger than 2 does not create multiple edges for white vertices of degree 2.

Since the graph G has neither self-loops nor multiple edges, by Fact 1, the embedding $\pi(G)$ has all its faces of size at least 3. Let G' be the graph and $\pi(G')$ be the embedding of G' after applying the above process on all white vertices in G. By the above discussion, the embedding $\pi(G')$ has genus g and all faces in $\pi(G')$ have size at least 3. We estimate the number m_{bb} of bb-edges in the graph G'. For each white vertex u of degree 2 with neighbors u_1 and u_2 in G', we associate the bb-edge $[u_1, u_2]$ with the two bw-edges $[u, u_1]$ and $[u, u_2]$. For each white vertex u of degree $d > 2$ with neighbors u_1, u_2, $\ldots$, u_d in G', for each $i = 1, 2, \ldots, d$ (here we take $u_{d+1} = u_1$), we associate the bb-edge $[u_i, u_{i+1}]$ that is on the boundary of the triangle face (u, u_i, u_{i+1}) with the bw-edge $[u, u_i]$. Note that each such bb-edge $[u_i, u_{i+1}]$ can be associated with at most two bw-edges because each edge can be on the boundaries of at most two faces. Moreover, the bb-edge $[u_i, u_{i+1}]$ cannot be associated with the two bw-edges incident on any degree-2 white vertex since u_i and u_{i+1} cannot be the neighbors of a degree-2 white vertex in G' (see the discussion in the last paragraph). Since every bw-edge must be incident on a white vertex, the above association shows that the number m_{bw} of bw-edges is at most twice of the number m_{bb} of bb-edges in G': $m_{bw} \leq 2m_{bb}$. Since the bw-edges in the graph G' are the same as those in the graph G, and the number of bb-edges in G is no more than that in G', we obtain

$$m \leq m_{bw} + m_{bb} \leq 3m_{bb} \tag{4.2}$$

Moreover, since each white vertex in G has degree at least 2, it is easy to see that the number n_w of white vertices in G is at most half the number m_{bw} of bw-edges in G. Thus,

$$n_w \leq m_{bw}/2 \leq m_{bb} \tag{4.3}$$

Recall that the embedding $\pi(G')$ has genus g and all faces in $\pi(G')$ have size at least 3. Now we remove all white vertices from the graph G' and from the embedding $\pi(G')$. Let the resulting graph and embedding be G'' and $\pi(G'')$, respectively. By Fact 3 and Fact 4, removing a white vertex neither changes the embedding genus nor creates faces of size less than 3. Thus, the embedding $\pi(G'')$ has genus g and all faces in $\pi(G'')$ have size at least 3. Note that the number of edges in G'' is equal to the number m_{bb} of bb-edges in G', and the number of vertices in G'' is equal to the number n_b of black vertices in G. Applying Lemma IV.4 to the graph G'', we get

$$m_{bb} \leq 6g + 3n_b - 6$$

Replacing m_{bb} by $6g+3n_b-6$ in relations (4.2) and (4.3), and noting that $n = n_w + n_b$ complete the proof of the lemma. $\square$

Now we are ready to prove the following theorem.

Theorem IV.6 *The Dominating Set problem on graphs of genus bounded by $g(n)$ is* *fixed parameter tractable if* $g(n) = n^{o(1)}$.

Proof. Since $g(n) = n^{o(1)}$, we can write $g(n) \leq n^{1/r(n)}$ for some nondecreasing and unbounded function $r(n)$. For an instance (G, k) of the Dominating Set problem, where the graph G has n vertices and genus g', we apply the algorithm **DS-FPT** in Figure 9.

```
ALGORITHM. DS-FPT
Input: a graph $G$ of $n$ vertices and an integer $k$
Output: decide if $G$ has a dominating set of $k$ vertices
1.  if $k \geq r(n)$ then solve the problem by enumerating all subsets of $k$ vertices in $G$; Stop;
2.  $k_0 = k$;   $D = \emptyset$;   $G_0 = G$;   color all vertices of $G_0$ black;
3.  while there is a black vertex $u$ of degree $d \leq 19$ in $G_0$ do
3.1.      make a $(d+1)$-way branch, each includes either $u$ or a neighbor of $u$ in $D$;
3.2.      remove the new vertex in $D$ from $G_0$, and color its neighbors in $G_0$ white;
3.3.      apply rules R1-R3 to make $G_0$ a reduced BW-graph;
3.4.      $k_0 = k_0 - 1$;
4.  if the graph $G_0$ has at most $78n^{1/k}$ vertices
4.1.      then find a B-dominating set of $k_0$ vertices in $G_0$ by enumerating all vertex subsets
                of $k_0$ vertices in $G_0$
4.2.      else Stop ("the graph $G$ has genus larger than $g(n)$");
```

Fig. 9. A parameterized algorithm for Dominating Set

Let $\overline{r}$ be the inverse function of the function $r(n)$ defined by $\overline{r}(p) = \min\{\, q \mid r(q) \geq p \,\}$. Then the function $\overline{r}$ is also nondecreasing and unbounded. In case $k \geq r(n)$, we have $\overline{r}(k) \geq n$. Thus, step 1 of the algorithm DS-FPT takes time $O(2^n) = O(2^{\overline{r}(k)})$.

Now suppose $k < r(n)$, step 3 repeatedly branches at a black vertex of degree bounded by 19 in the reduced BW-graph G_0. The search tree size $T(k)$ of step 3 thus satisfies the recurrence relation

$$T(k) \leq 20 \cdot T(k-1)$$

which has a solution $T(k) = O(20^k)$.

At the end of step 3, all black vertices in the reduced BW-graph G_0 have degree at least 20. Suppose at this point, the number of edges, the number of vertices, and the number of black vertices in G_0 are m_0, n_0 and n_b, respectively. Since $2m_0$ is equal to the sum of total vertex degrees in G_0, we have $2m_0 \geq 20n_b$. By Lemma IV.5(a), we also have $m_0 \leq 9n_b + 18g' - 18$ (note that the genus of the reduced BW-graph G_0 cannot be larger than the genus g' of the original graph G). Combining these two relations, we get $n_b \leq 18g' - 18$. By Lemma IV.5(b), we have $n_0 \leq 4n_b + 6g' - 6$. Thus

$$n_0 \leq 4n_b + 6g' - 6 \leq 78g' - 78 < 78g'$$

Thus, if $g' \leq g(n) \leq n^{1/r(n)} < n^{1/k}$ (note $k < r(n)$), then the number n_0 of vertices in the graph G_0 must be bounded by $78n^{1/k}$. In this case, step 4.1 solves the problem in time $O(n_0^{k_0+1}) = O((n^{1/k})^k) = O(n)$. On the other hand, if G_0 has more than $78n^{1/k}$ vertices, then step 4.2 concludes correctly that the genus of the input graph G is larger than $g(n)$.

In conclusion, the algorithm **DS-FPT** solves the Dominating Set problem on graphs of genus bounded by $g(n)$ in time $O(2^{\bar{r}(k)} + 20^k + n)$, and the problem is fixed parameter tractable. $\square$

We point out that the techniques used in Theorem IV.6 are simpler, more uniform, and derive much stronger results compared to the previous research, which was only valid for graphs of genus bounded by a constant [33]. Also, similarly to the algorithm **IS-FPT**, the algorithm **DS-FPT** does not have to know whether the input graph has minimum genus bounded by $g(n)$. For any graph of minimum genus bounded by $g(n)$, the algorithm will definitely derive a correct conclusion.

Theorem IV.7 *The Dominating Set problem on graphs of genus bounded by $g(n)$ is $W[2]$-complete if $g(n) = n^{\Omega(1)}$.*

PROOF. Since Dominating Set is $W[2]$-complete [11], it will suffice to show that Dominating Set on general graphs is fpt-reducible to the problem on graphs of genus bounded by $g(n)$. Since $g(n) = n^{\Omega(1)}$, we can assume that $g(n) \geq n^c$, where c is a fixed constant.

Let G_1 be an arbitrary graph of n_1 vertices and genus g_1. As we indicated in the proof of Theorem IV.2, $g_1 \leq n_1^2$. We construct a new graph G_2, which is the graph G_1, plus $n_1^{2/c} - n_1$ new vertices u, v, and v_i, $i = 1, 2, \ldots, n_1^{2/c} - n_1 - 2$, where u has degree 2 and is connected to the vertex v and to an arbitrary vertex in the graph G_1, and $[v, v_i]$, $i = 1, 2, \ldots, n_1^{2/c} - n_1 - 2$, make a star centered at v. It is fairly easy to verify that the graph G_2 has $n_2 = n_1^{2/c}$ vertices and genus $g_2 = g_1$, and that the graph G_1 has a dominating set of k_1 vertices if and only if the graph G_2 has a dominating set of $k_2 = k_1 + 1$ vertices. Since c is a constant, the reduction from (G_1, k_1) to (G_2, k_2) is an fpt-reduction. Moreover, since $g_2 = g_1 \leq n_1^2$, we have $g_2 \leq n_2^c \leq g(n_2)$. Therefore, (G_2, k_2) is an instance for Dominating Set on graphs of genus bounded by $g(n)$. This reduction proves that Dominating Set on graphs of genus bounded by $g(n)$ is $W[2]$-complete. □

Combining Theorem IV.6 and Theorem IV.7, we derive the following tight result.

Corollary IV.8 *Assuming $FPT \neq W[2]$, the Dominating Set problem on graphs of genus bounded by $g(n)$ is fixed parameter tractable if and only if $g(n) = n^{o(1)}$.*

C. Genus and Subexponential Time Complexity

We say that a graph problem is solvable in *sublinear exponential time* (or shortly *subexponential time*) if it can be solved in time $2^{o(n)}$ on graphs of n vertices. Very few NP-hard graph problems are known to be solvable in subexponential time. Lipton and

Tarjan used their planar graph separator theorem to show that a class of NP-hard planar graph problems, including Vertex Cover, Independent Set, and Dominating Set, are solvable in subexponential time [54]. They also described how their results can be extended to graphs of constant genus [54]. Recently, deriving lower bounds on the precise complexity of NP-hard problems has been attracting more and more attention [14, 15, 60, 61]. In particular, Impagliazzo, Paturi, and Zane [15] introduced the concept of SERF-reduction and showed that many well-known NP-hard problems are SERF-complete for the class SNP [15, 67]. This implies that if any of these problems is solvable in subexponential time, then so are all problems in the class SNP, a consequence that seems quite unlikely.

In this section, we demonstrate how graph genus affects the subexponential time computability for Vertex Cover, Independent Set, and Dominating Set. Our algorithmic results in this section extend Lipton and Tarjan's results on planar graphs and graphs of constant genus [54], and our lower bound results refine Impagliazzo, Paturi, and Zane's results on general graphs [15].

Proposition IV.9 ([68]) *Let G be a graph of n vertices and genus g. There is a linear time algorithm that partitions the vertices of G into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, $|A|, |B| \leq n/2$, and $|C| \leq c_0\sqrt{(g+1)n}$, where c_0 is a fixed constant.*

Theorem IV.10 *Vertex Cover, Independent Set, and Dominating Set on graphs of genus bounded by $g(n)$ are solvable in subexponential time if $g(n) = o(n)$.*

Proof. We first give a detailed description of our proof for Dominating Set. The idea is quite simple: we use Proposition IV.9 to partition the vertices of a given graph G into the three sets A, B, and C, and enumerate all possible situations for the set C.

Each fixed situation for the set C splits the graph G into two separated subgraphs, induced essentially by the vertex sets A and B, respectively. Thus, we can recursively work on the two subgraphs independently. However, this must be done with care. In particular, in a given situation for the set C, if a vertex u in C is assigned to be not in the dominating set and u is not adjacent to any vertex in C that is assigned to be in the dominating set, then the vertex u must remain in the graph and a vertex in A or B and adjacent to u must be included in the dominating set in a later stage.

Thus, assuming recursively that a partial dominating set D has been constructed, our recursive algorithm classifies the vertices in the remaining graph G into five groups:

1. dominating vertices, which are already included in the current D;
2. dominated vertices, which should not be in D and are adjacent to vertices in the current D;
3. white vertices, which are adjacent to vertices in the current D but are not yet decided whether to be in D;
4. black vertices, which are not adjacent to any vertices in the current D and are also not yet decided whether to be in D;
5. red vertices, which should not be in D but are not yet adjacent to any vertices in the current D.

The dominating vertices and dominated vertices will be removed from the graph. Thus, the remaining graph G consists of only black, red, and white vertices (initially, $D = \emptyset$ and all vertices in G are black). Such a graph G will be called a *BWR-graph*. A *BW-dominating set* D' in the BWR-graph G is a set of black and white vertices in G such that every vertex in G is either in D' or adjacent to a vertex in D' (thus, a minimum BW-dominating set for the initial graph will be a regular minimum dominating set for the graph). To construct a minimum BW-dominating set for the BWR-graph G, we use Proposition IV.9 to partition the vertices of G into the three vertex subsets A, B, and C. Then we consider all possible assignments on the vertices in the set C. Each vertex u in C has the following possible assignments:

- u is a white vertex. Then either u is in D or u is not in D;
- u is a red vertex. Then u must be dominated by a vertex in either C, or A, or B;
- u is a black vertex. Then either u is in D, or u is not in D, and hence must be dominated by a vertex in either C, or A, or B.

An assignment to the vertices in C can be as follows: each white vertex is assigned either "in-D" or "not-in-D", each red vertex is assigned either "in-A" or "in-B", and each black vertex is assigned either "in-D", "in-A", or "in-B". After this assignment, a white vertex will become either a dominating vertex (if it is "in-D") or a dominated vertex (if it is "not-in-D"); a red vertex adjacent to an "in-D" vertex in C will become a dominated vertex (in this case, the assignment to the red vertex is ignored); a red vertex not adjacent to any "in-D" vertex in C will become a red vertex and will be added to the set A or B (depending on whether it is an "in-A" or "in-B" vertex); an "in-D" black vertex will become a dominating vertex; a black vertex whose status is either "in-A" or "in-B" and is adjacent to an "in-D" vertex in C will become a dominated vertex; finally, an "in-A" black vertex (resp. an "in-B" black vertex) not adjacent to any "in-D" vertex in C will become a red vertex and will be added to the set A (resp. B).

Let the subgraphs induced by the updated vertex sets A and B be G_A and G_B, respectively (note that now A and B may contain some vertices that were originally in C). We then recursively work on the subgraphs G_A and G_B. The algorithm is formally presented in Figure 10.

```
ALGORITHM. DS-solver
Input: a BWR-graph $G$ of $n$ vertices, and a bound $b_0$
Output: a minimum BR-dominating set $D$ of $G$
1. if $\sqrt{n} < 6b_0$ then solve the problem by a brute force method; Stop;
2. partition the vertices of $G$ into the subsets $A$, $B$, $C$, as described in Proposition IV.9;
3. if $|C| > b_0\sqrt{n}$ then Stop("the genus exceeds the bound");
4. for each assignment to the vertices in $C$ do
       let $D$ be the set of vertices in $C$ that are assigned "in-$D$";
       update the graph $G$ and the sets $A$ and $B$;
       construct the subgraphs $G_A$ and $G_B$;
       recursively construct the minimum BR-dominating sets $D_A$ for $G_A$ and $D_B$ for $G_B$;
       $D = D \cup D_A \cup D_B$;
5. output the smallest BR-dominating set constructed in step 4.
```

Fig. 10. An algorithm solving Dominating Set

We analyze the algorithm. Suppose the original input graph G_0 has n_0 vertices. Set $b_0 = c_0\sqrt{g(n_0)+1}$, where c_0 is the constant given in Proposition IV.9 (the bound b_0 is fixed for all recursive calls to the algorithm **DS-Solver**). Suppose that the input to the algorithm **DS-Solver** is a BWR-graph G of n vertices. If $\sqrt{n} < 6b_0$, then $n < 36c_0^2(g(n_0)+1) = O(g(n_0))$, and a brute force method can construct a minimum BR-dominating set for G in time $O(3^n) = O(3^{O(g(n_0))})$. If $|C| > b_0\sqrt{n}$, then C would contain more than $c_0\sqrt{(g(n_0)+1)n}$ vertices. By Proposition IV.9, the graph G would have genus larger than $g(n_0)$, which implies that the original input graph G_0 has genus larger than $g(n_0)$ (since G is a subgraph of G_0). Thus, the algorithm stops correctly.

Thus, we have $\sqrt{n} \geq 6b_0$ and $|C| \leq b_0\sqrt{n}$. Since each vertex in C can get at most 3 different assignments, the total number of different assignments to the set C is bounded by $3^{|C|} \leq 3^{b_0\sqrt{n}}$. Since originally, $|A|, |B| \leq n/2$, and the updated sets A and B are the original sets A and B plus some vertices in C, each of the subgraphs G_A and G_B contains at most $n/2 + b_0\sqrt{n} \leq 2n/3$ vertices (note that $b_0 \leq \sqrt{n}/6$). This gives the following recurrence relation for the time complexity $T(n)$ of the algorithm **DS-Solver**:

$$\begin{aligned} T(n) \leq 3^{b_0\sqrt{n}} \cdot 2T(2n/3) \leq 3^{b_0\sqrt{n}+1}T(2n/3) & \qquad \text{if } \sqrt{n} \geq 6b_0 \\ T(n) = O(3^{O(g(n_0))}) & \qquad \text{if } \sqrt{n} < 6b_0 \end{aligned}$$

Solving this recurrence relation, we get $T(n) = O(3^{O(b_0\sqrt{n}+g(n_0))})$. In particular, if we let $n = n_0$ and replace b_0 by $c_0\sqrt{g(n_0)+1}$, we get

$$T(n_0) = O(3^{O(c_0\sqrt{g(n_0)+1}\cdot\sqrt{n_0}+g(n_0))}) = 3^{O(\sqrt{n_0 g(n_0)}+g(n_0))} \tag{4.4}$$

Thus, if $g(n_0) = o(n_0)$, then $T(n_0) = 2^{o(n_0)}$, and the algorithm **DS-Solver** solves the Dominating Set problem in subexponential time.

The subexponential time algorithms for Vertex Cover and Independent Set are similar, and actually simpler. For example, for Vertex Cover, once we partition the input graph into three parts A, B, and C, each vertex u in C has only two possibilities: either in or not in the minimum vertex cover W. In case u is in W, we simply remove u from the graph; while in case u is not in W, all neighbors of u are forced to be in W, thus all neighbors of u, as well as u itself, can be removed from the graph. Therefore, no vertices in C will be added to the sets A and B, and each of the induced subgraphs G_A and G_B will have at most $n/2$ vertices. This fact will simplify the analysis of the algorithm to derive the subexponential time bound. We leave the detailed verification to the interested readers. $\square$

Again we point out that our subexponential time algorithms for Dominating Set, Vertex Cover, and Independent Set work correctly without needing to know the precise genus value of the input graph. The algorithms either report correctly that the genus of the input graph exceeds the designated bound $g(n)$, or construct an optimal solution to the input graph.

Remark. After the publication of a preliminary version [31] of the current chapter in 2003, there has been some further progress in this direction. Demaine *et al.* [55] developed an algorithm of running time $2^{O(g\sqrt{k}+g^2)}n^{O(1)}$ for the parameterized Dominating Set problem on graphs of genus bounded by g, which was further improved by Fomin and Thilikos [56] who presented an algorithm of running time $2^{O(\sqrt{kg}+g)}+n^{O(1)}$. Compared to the algorithm in [55], our algorithm in Theorem IV.10 is faster when the graph genus g is $\Omega(\sqrt{n})$. Compared to the algorithm in [56], the running time of our algorithm (see Equality (4.4)) is of the same order as that of the algorithm in [56] for the general version (i.e., the non-parameterized version) of the Dominating Set problem (since the parameter k can be of order $\Theta(n)$). Moreover, our algorithm seems much simpler (the algorithm in [56] uses the techniques of graph representativity and graph branch decomposition).

Theorem IV.11 *For any function $g(n) = \Omega(n)$, if any of Vertex Cover, Independent Set, and Dominating Set on graphs of genus bounded by $g(n)$ can be solved in subexponential time, then all problems in the class SNP can be solved in subexponential time.*

Proof. Since $g(n) = \Omega(n)$, we assume $g(n) \geq cn$, where c is a fixed constant. Johnson and Szegedy [16] have shown that if Independent Set on graphs of degree bounded by 3 is solvable in subexponential time then so is Independent Set on general graphs, which, according to Impagliazzo, Paturi, and Zane [15], would imply that all problems in the class SNP are solvable in subexponential time. Therefore, for Independent Set, it suffices to show that the problem on graphs of degree bounded by 3 is reducible to the problem on graphs of genus bounded by $g(n)$ via a reduction that preserves the order of the number of vertices.

Let n_1, m_1, and g_1 be the number of vertices, the number of edges, and the genus of a graph G_1 of degree bounded by 3. Then $m_1 \leq 3n_1/2$, and by the Euler Polyhedral Equation [58], $g_1 \leq (m_1 - n_1 + 1)/2 \leq (n_1 + 2)/4 \leq n_1/3$ for $n_1 \geq 6$. If $c \geq 1/3$, then G_1 is already a graph of genus bounded by $cn_1 \leq g(n_1)$. Thus, we assume $c < 1/3$. We perform the following operation on the graph G_1. Pick any edge in G_1, and subdivide the edge by two degree-2 vertices. The resulting graph G' has $n_1 + 2$ vertices and the same genus g_1. Moreover, it can be proved [12, 69] that from any maximum independent set of G', a maximum independent set of G_1 can be constructed in linear time. Therefore, if we apply this edge subdivision operation $\lceil n_1/(6c) - n_1/2 \rceil$ times on the graph G_1, we get a graph G_2 of n_2 vertices and genus $g_2 = g_1$, where $n_1/(3c) \leq n_2 \leq (3c+1)n_1/(3c)$. Now since $g_2 = g_1 \leq n_1/3 \leq cn_2$, the graph G_2 of n_2 vertices has genus bounded by $g(n_2)$. The reduction is completed by observing that $n_2 = O(n_1)$.

The theorem also holds for Vertex Cover since Independent Set can be reduced to Vertex Cover using the same graph [6]. For Dominating Set, the theorem follows from the following facts: (1) Vertex Cover on graphs of degree bounded by 3 can be reduced to Dominating Set on graphs of degree bounded by 6 [6]; and (2) subdividing an edge by three degree-2 vertices increases the minimum dominating set size by 1 [69] and does not change the graph genus. With these facts, the proof proceeds in a similar fashion to that for Independent Set. We leave the details to interested readers.

$\square$

The class SNP [67] contains many well-known NP-hard problems, including k-SAT, k-Colorability, k-Set Cover, Vertex Cover, and Independent Set [15]. It is commonly believed that it is unlikely that all problems in SNP are solvable in subexponential time. Based on this, and combining Theorem IV.10 and Theorem IV.11, we have the following tight results.

Corollary IV.12 *Assuming that not all the problems in SNP are solvable in subexponential time, the Vertex Cover, Independent Set, and Dominating Set problems on graphs of genus bounded by $g(n)$ are solvable in subexponential time if and only if $g(n) = o(n)$.*

D. Genus and Approximability

We briefly review the related concepts and refer the readers to [2, 6] for more details. An *optimization problem* Q is either a *maximization* or a *minimization* problem. Each instance x of Q is associated with a set of *solutions* and each solution y for x is associated with a value $f(x, y)$. For a given instance x in Q, the objective is to find a solution with the maximum value $\max(x)$ (if Q is a maximization problem) or the minimum value $\min(x)$ (if Q is a minimization problem). An *approximation algorithm* A for Q is an algorithm that for each instance x of Q constructs a solution $A(x)$ for x. We say that the *approximation ratio* of the algorithm A is bounded by r if for all instances x of Q, we have $\max(x)/f(x, A(x)) \leq r$ (if Q is a maximization problem) or $f(x, A(x))/\min(x) \leq r$ (if Q is a minimization problem). We say that an optimization problem Q has a *polynomial time approximation scheme*, shortly *PTAS*, if for any constant $\epsilon > 0$, the problem Q has a polynomial time approximation algorithm whose approximation ratio is bounded by $1+\epsilon$. It is well-known that Vertex Cover, Independent Set, and Dominating Set on planar graphs have PTAS [53, 54].

Proposition IV.13 ([68]) *There is an $O(n \log g)$ time algorithm that for a given graph G of n vertices and genus g constructs a subset Z of at most $c\sqrt{gn \log g}$ vertices, where c is a fixed constant, such that removing the vertices in Z from G results in a planar graph.*

The algorithm in Proposition IV.13 does not need to know the genus of the input graph [68].

Theorem IV.14 *The Independent Set problem on graphs of genus bounded by $g(n)$ has a PTAS if $g(n) = o(n/\log n)$.*

Proof. Let $g(n) \leq n/(r(n)\log n)$, where $r(n)$ is a nondecreasing and unbounded function. Our PTAS for Independent Set works as follows: for a given graph G of n vertices, we use the algorithm in Proposition IV.13 to construct the vertex subset Z (this can be done in time $O(n \log n)$ even when the genus of G is larger than $g(n)$). If the number z_0 of vertices in Z is larger than $c\sqrt{g(n)n\log g(n)})$, then we know that the input graph G has genus larger than $g(n)$ and we stop. Otherwise, the graph G_1 obtained by deleting the vertices in Z from the graph G is a planar graph. We apply any known PTAS algorithm (e.g., those given in [53, 54]) to construct an independent set I_1 for the graph G_1. We simply output I_1 as a solution to the original graph G.

It is obvious that this is a polynomial time approximation algorithm for Independent Set on graphs of genus bounded by $g(n)$. What left is to analyze the approximation ratio of the algorithm. Because $g(n) \leq n/(r(n)\log n))$, the number of vertices z_0 in Z is such that $z_0 \leq c\sqrt{g(n)n\log g(n)}) \leq cn/\sqrt{r(n)}$. Let $n_1 = n - z_0$ be the number of vertices in the graph G_1. Let α and α_1 be the sizes of a maximum independent set in the graphs G and G_1, respectively. Then $\alpha_1 \leq \alpha \leq \alpha_1 + z_0$. Because G_1 is a planar graph, by the Four-Color theorem [58], $\alpha_1 \geq n_1/4$.

Let $\alpha_1' = |I_1|$. Since the independent set I_1 is constructed by a PTAS on the planar graph G_1, $\alpha_1/\alpha_1' \leq 1 + \epsilon$, where ϵ is the given error bound. Since the function $r(n)$ is nondecreasing and unbounded, there is a constant N_0 such that when $n \geq N_0$, we have

$$\frac{c}{4\sqrt{r(n)}} \leq \frac{1}{8} \qquad \text{and} \qquad \frac{8c(1+\epsilon)}{\sqrt{r(n)}} \leq \epsilon \tag{4.5}$$

From the first inequality, we get

$$\begin{aligned} \alpha_1' &\geq \frac{\alpha_1}{1+\epsilon} \geq \frac{n_1}{4(1+\epsilon)} = \frac{n-z_0}{4(1+\epsilon)} \geq \frac{n - cn/\sqrt{r(n)}}{4(1+\epsilon)} \\ &= n \cdot \left(\frac{1}{4(1+\epsilon)} - \frac{c}{4(1+\epsilon)\sqrt{r(n)}} \right) \geq \frac{n}{8(1+\epsilon)} \end{aligned} \tag{4.6}$$

Since $\alpha \leq \alpha_1 + z_0 \leq (1+\epsilon)\alpha_1' + cn/\sqrt{r(n)}$, combining this with (4.5) and (4.6), we get

$$\frac{\alpha}{\alpha_1'} \leq 1 + \epsilon + \frac{cn}{\alpha_1'\sqrt{r(n)}} \leq 1 + \epsilon + \frac{8cn(1+\epsilon)}{n\sqrt{r(n)}} \leq 1 + 2\epsilon$$

Thus, the algorithm is a PTAS for Independent Set on graphs of genus bounded by $g(n)$. □

Again our PTAS for Independent Set does not need to know whether the input graph meets the given genus bound.

Theorem IV.15 *Assuming P ≠ NP, then Independent Set on graphs of genus bounded by $g(n)$ has no PTAS if $g(n) = \Omega(n)$.*

Proof. The proof uses techniques similar to those in Theorem IV.11, so we only give an outline of it. It is known that Independent Set on graphs of bounded degree is APX-complete [2], which means that a PTAS for it would imply P = NP [70]. Now a graph G_1 of n_1 vertices and of bounded degree has its genus bounded by $O(n_1)$. We can increase the number of vertices in G_1 without changing the graph genus by subdividing the edges in G_1 by degree-2 vertices (see the proof of Theorem IV.11). This will give a graph G_2 of n_2 vertices whose genus is bounded by $g(n_2)$ (note that $g(n) \geq cn$ for some constant c), and a PTAS for the graph G_2 would imply a PTAS for the graph G_1. In consequence, a PTAS for Independent Set on graphs of genus bounded by $g(n)$ would imply a PTAS for the same problem on graphs of bounded degree, which would imply that P = NP. $\square$

Theorem IV.14 seems unlikely to hold for Vertex Cover and Dominating Set. In fact, we can prove the following theorem.

Theorem IV.16 *Unless P = NP, Vertex Cover and Dominating Set on graphs of genus bounded by $g(n)$ have no PTAS if $g(n) = n^{\Omega(1)}$.*

Proof. It is known that Vertex Cover and Dominating Set on general graphs have no PTAS unless P = NP [70, 67]. Thus, it suffices to show how these problems on general graphs can be reduced to the ones on graphs of genus bounded by $g(n) = n^{\Omega(1)}$. The proof is very similar to that for Theorem IV.7, thus we only give an outline of it. Consider the Dominating Set problem. For a given general graph G_1 of n_1 vertices, by attaching to G_1 a very large star, we can construct a new graph G_2 of n_2 vertices, without changing the graph genus, such that the genus of the graph G_2 is bounded by $g(n_2)$, and that the domination numbers of the graphs G_1 and G_2 differ by exactly 1. Now a PTAS for the graph G_2 would imply a PTAS for the graph G_1. The theorem for Vertex Cover can be proved using a similar construction. $\square$

On the other hand, we can derive results similar to Theorem IV.14 for Vertex Cover and Dominating Set on "kernelized" graphs. Polynomial time kernelization algorithms have become an interesting topic in the recent research on NP-hard problems [71, 12, 56]. It has been demonstrated [72] that improvement on approximating Vertex Cover and Dominating Set on kernelized graphs will directly imply the same improvement on approximating the problems on general graphs. In the following, we discuss the impact of graph genus on the approximability of Vertex Cover and Dominating Set on kernelized graphs.

For an arbitrary graph G, the kernelization algorithms construct a kernelized graph G', where a vertex cover C' for the graph G' gives directly a vertex cover C for the graph G that preserves the approximation ratio (that is, the ratio of C to an optimal solution of G is not worse than the ratio of C' to an optimal solution of G').

Theorem IV.17 *The Vertex Cover problem on kernelized graphs of genus bounded by $g_1(n)$ has a PTAS if $g_1(n) = o(n/\log n)$. On the other hand, unless P = NP, the Vertex Cover problem on kernelized graphs of genus bounded by $g_2(n)$ has no PTAS if $g_2(n) = \Omega(n)$.*

PROOF. The development of a PTAS for Vertex Cover on graphs of genus bounded by $g_1(n) = o(n/\log n)$ is very similar to that for the PTAS for Independent Set given in Theorem IV.14, except that for Independent Set in Theorem IV.14, we used Four-Color theorem to derive a linear lower bound on the size of maximum independent sets for planar graphs, while for Vertex Cover on kernelized graphs, the linear lower bound on the size of minimum vertex covers comes directly from the fact that the input graph is kernelized. To prove that Vertex Cover has no PTAS on kernelized graphs of genus bounded by $g_2(n) = \Omega(n)$, we use the techniques given in the proof of Theorem V.21, by observing that a graph obtained by applying the operations given in Theorem IV.11 (i.e., subdividing an edge by two degree-2 vertices [69]) on a kernelized graph is also kernelized. We leave the detailed verification to the interested readers. $\square$

Very recently, a kernelization algorithm for Dominating Set has been proposed. For a given graph G, let $\delta(G)$ be the size of a minimum dominating set in the graph

G and recall that $\gamma_{\min}(G)$ denotes the minimum genus of the graph G. Formin and Thilikos [56] proposed a polynomial time algorithm that reduces a given graph G to a graph G' such that $\delta(G) = \delta(G')$, and such that the number of vertices of G' is bounded by $c_0(\delta(G') + \gamma_{\min}(G'))$, where $c_0 > 4$ is a constant. Based on this result, we can introduce the following definition: we say that a graph G is *kernelized for the Dominating Set problem* if the number of vertices in G is bounded by $c_0(\delta(G) + \gamma_{\min}(G))$, where c_0 is the constant given in [56].

Theorem IV.18 *The Dominating Set problem on kernelized graphs of genus bounded by $g_1(n)$ has a PTAS if $g_1(n) = o(n/\log n)$. On the other hand, unless P = NP, the Dominating Set problem on kernelized graphs of genus bounded by $g_2(n)$ has no PTAS if $g_2(n) = \Omega(n)$.*

Proof. We only sketch the proof, which is similar to that for Theorem IV.17. We leave the detailed verification to the interested reader.

The PTAS for Dominating Set on kernelized graphs of genus bounded by $g_1(n) = o(n/\log n)$ is obtained in a similar way to the PTAS for Vertex Cover given in Theorem IV.17, with the lower bound on the size of minimum dominating sets coming from the kernelization. To prove that Dominating Set has no PTAS on kernelized graphs of genus bounded by $g_2(n) = \Omega(n)$, we note that graphs of degree bounded by 3 are necessarily kernelized since the size of a minimum dominating set in such a graph is at least $n/4$ – each vertex can dominate at most 3 other vertices in the graph. Moreover, the genus of such a graph is bounded by $O(n)$ [58]. Therefore, the assumed PTAS for Dominating Set on graphs of genus bounded by $g_2(n)$ would imply a PTAS for Dominating Set on graphs of degree bounded by 3, which is APX-complete [2]. This, in consequence, would imply P = NP. □

E. Comments

We have demonstrated how graph genus affects the computational complexity of the well-known NP-hard problems Vertex Cover, Independent Set, and Dominating Set in terms of the following complexity measures: the fixed parameter tractability, the subexponential time computability, and the polynomial time approximability. In most cases, we were able to derive a precise genus threshold that uniquely determines the computational complexity of the problems in terms of the complexity measures. Our algorithmic results significantly extend the previous research on the problems on planar graphs and on graphs of constant genus, while our complexity results refine the previous results on the problems and identify the "hardest graph instances" for the problems. It should be easy to see that our techniques and results can be extended to other NP-hard graph problems.

It is NP-hard to determine the minimum genus of a given graph [62]. However, it is interesting to point out that all the algorithms developed in this chapter work correctly without needing to know whether the input graph exceeds the designated genus bound. Our algorithms either report correctly that the input graph exceeds the designated genus bound, or solve the problems correctly for the given graph. Our techniques seem to be useful for the study of other computational problems related to graph genus.

CHAPTER V

COMPUTATIONAL LOWER BOUNDS VIA PARAMETERIZED COMPLEXITY

In this chapter, we develop new techniques for deriving very strong computational lower bounds for a class of well-known NP-hard problems including Weighted Satisfiability, Dominating Set, Hitting Set, Set Cover, Clique, and Independent Set. For example, although a trivial enumeration can easily test in time $O(n^k)$ if a given graph of n vertices has a clique of size k, we prove that unless an unlikely collapse occurs in parameterized complexity theory, the problem is not solvable in time $f(k)n^{o(k)}$ for *any* function f, even if we restrict the parameter values to be bounded by an arbitrarily small function of n. We further extended our techniques to derive computational lower bounds on polynomial time approximation schemes for NP-hard optimization problems. For example, we prove that the NP-hard Distinguishing Substring Selection problem, for which a polynomial time approximation scheme has been recently developed, has no polynomial time approximation schemes of running time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f unless an unlikely collapse occurs in parameterized complexity theory.

A. A New Approach to Proving Lower Bounds

The $W[1]$-hardness of a parameterized problem implies that any algorithm of running time $O(n^h)$ solving the problem *must* have h a function of the parameter k. However, this does not completely exclude the possibility that the problem may become feasible for small values of the parameter k. For instance, if the problem is solvable by an algorithm running in time $O(n^{\log\log k})$, then such an algorithm is still feasible for moderately small values of k.[1]

Based on the framework of parameterized complexity theory, we develop new techniques and derive much stronger computational lower bounds for a class of well-known NP-hard problems. In particular, we answer the above mentioned questions completely. We start by proving computational lower bounds for a class of Satisfiability problems, and then extend the lower bound results to other well-known NP-hard problems by introducing the concept of *linear fpt-reductions*. In particular, we consider two classes of parameterized problems: Class A which includes Weighted CNF SAT, Dominating Set, Hitting Set, and Set Cover, and Class B which includes Weighted CNF q-SAT for any constant $q \geq 2$, Clique, and Independent Set. We prove that (1) unless $W[1] = FPT$, no problem in Class A can be solved in time $f(k)n^{o(k)}m^{O(1)}$ for *any* function f, where n is the size of the search space from which the k elements are selected and m is the input length; and (2) unless all search problems in the syntactic class SNP introduced by Papadimitriou and Yannakakis [67] are solvable in subexponential time, no problem in Class B can be solved in time $f(k)m^{o(k)}$ for *any* function f, where m is the input length. These results remain true even if we bound the parameter values by an arbitrarily small nondecreasing and unbounded function. Moreover, under the same assumptions, we prove that even if we restrict the parameter values k to be of the order $\Theta(\mu(n))$ for *any* reasonable function μ, no problem in Class A can be solved in time $n^{o(k)}m^{O(1)}$ and no problem in Class B can be solved in time $m^{o(k)}$. These are very significant improvements over the results given in [60]: the results in [60] establish lower bounds because of *a particular* value of the parameter, while the results in the current chapter under the same assumptions

[1]A question that might come to mind is whether such a $W[1]$-hard problem exists. The answer is affirmative: by re-defining the parameter, it is not difficult to construct $W[1]$-hard problems that are solvable in time $O(n^{\log\log k})$.

claim the lower bounds for essentially *every* value of the parameter.

Note that each of the problems in Class A (resp. Class B) can be solved by a trivial algorithm of running time $cn^k m$ (resp. cm^k), where c is an absolute constant, which simply enumerates all possible subsets of k elements in the search space. Much research has tended to seek new approaches to improve this trivial upper bound. One of the common approaches is to apply a more careful branch-and-bound search process trying to optimize the manipulation of local structures before each branch [73, 74, 12, 75, 76]. Continuously improved algorithms for these problems have been developed based on improved local structure manipulations (for example, see [44, 46, 28, 45] on the progress for the Independent Set problem). It has even been proposed to automate the manipulation of local structures [47, 77] in order to further improve the computational time.

Our results above, however, provide strong evidence that the power of this approach is quite limited in principle. The lower bounds $f(k)n^{\Omega(k)}p(m)$ and $f(k)m^{\Omega(k)}$ for any function f and any polynomial p mentioned above indicate that *no* local structure manipulation running in polynomial time or in time depending only on the value k will obviate the need for exhaustive enumerations.

Our techniques have also enabled us to derive lower bounds on the computational time of polynomial time approximation schemes (PTAS) for certain NP-hard problems. We pick the Distinguishing Substring Selection problem (DSSP) as an example, for which a PTAS was recently developed [78, 79]. Gramm et al. [80] showed that the parameterized DSSP problem is $W[1]$-hard, thus excluding the possibility that DSSP has a PTAS of running time $f(1/\epsilon)n^{O(1)}$ for any function f. We prove a much stronger result. We first show that the Dominating Set problem can be linearly fpt-reduced to the DSSP problem, thus proving that the parameterized DSSP problem is $W[2]$-hard (improving the result in [80]). We then show how this lower bound on parameterized complexity can be transformed into a lower bound on the computational complexity for any PTAS for the problem. More specifically, we prove that unless all search problems in SNP are solvable in subexponential time, the DSSP problem has no PTAS of running time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f. This essentially excludes the possibility that the DSSP problem has a practically efficient PTAS even for moderate values of the error bound ϵ. To the authors' knowledge, this is the first time a specific lower bound has been derived on the running time of a PTAS for an NP-hard problem.

In this chapter, we always assume that complexity functions are "nice" with both domain and range being non-negative integers and the values of the functions and their inverses can be easily computed. For two functions f and g, we write $f(n) = o(g(n))$ if there is a nondecreasing and unbounded function λ such that $f(n) \leq g(n)/\lambda(n)$. A function f is *subexponential* if $f(n) = 2^{o(n)}$.

B. Satisfiability and Weighted Satisfiability

In this section, we present two lemmas that show how a general satisfiability problem is transformed into a weighted satisfiability problem. One lemma is on circuits of bounded depth and the other lemma is on CNF formulas.

A *circuit* C of n input variables is a directed acyclic graph. The nodes of in-degree 0 are the *input gates*, each labelled uniquely either by a *positive literal* x_i or by a *negative literal* $\overline{x}_i$, $1 \leq i \leq n$. All other gates are either AND or OR gates. A special gate of out-degree 0 is designated as the *output gate*. The *size* of C is the number of gates in C, and the *depth* of C is the length of the longest path in C from an input gate to the output gate. A circuit is *monotone* (resp. *antimonotone*) if all its input gates are labelled by positive literals (resp. negative literals). A circuit represents a Boolean function in a natural way. We say that a truth assignment τ to the input variables of C *satisfies* a gate g in C if τ makes the gate g have the value 1, and that τ

satisfies the circuit C if τ satisfies the output gate of C. The *weight* of an assignment τ is the number of variables assigned the value 1 by τ.

A circuit C is a Π_t*-circuit* if its output gate is an AND gate and it has depth t. Using the results in [81], a Π_t-circuit C can be re-structured into an equivalent Π_t-circuit C' with size increased at most quadratically such that (1) C' has $t+1$ levels and each edge in C' only goes from a level to the next level; (2) the circuit C' has the same monotonicity and the same set of input variables; (3) level 0 of C' consists of all input gates and level t of C' consists of a single output gate; and (4) AND and OR gates in C' are organized into t alternating levels. Thus, without loss of generality, we will implicitly assume that Π_t-circuits are in this levelled form.

The Satisfiability problem on Π_t-circuits, abbreviated SAT[t], is to determine if a given Π_t-circuit C has a satisfying assignment. The parameterized problem Weighted Satisfiability on Π_t-circuits, abbreviated WCS[t], is to determine for a given pair (C, k), where C is a Π_t-circuit and k is an integer, if C has a satisfying assignment of weight k. The Weighted Monotone Satisfiability (resp. Weighted Antimonotone Satisfiability) problem on Π_t-circuits, abbreviated WCS$^+[t]$ (resp. WCS$^-[t]$) is defined similarly to WCS[t] with the exception that the circuit C is required to be monotone (resp. antimonotone). It is known that for each even integer $t \geq 2$, WCS$^+[t]$ is $W[t]$-complete, and for each odd integer $t \geq 2$, WCS$^-[t]$ is $W[t]$-complete. To simplify our statements, we will denote by WCS$^*[t]$ the problem WCS$^+[t]$ if t is even and the problem WCS$^-[t]$ if t is odd.

Lemma V.1 *Let $t \geq 2$ be an integer. There is an algorithm A_1 that, for a given integer $r > 0$, transforms each Π_t-circuit C_1 of n_1 input variables and size m_1 into an instance (C_2, k) of WCS*$^*[t]$*, where $k = \lceil n_1/r \rceil$ and the Π_t-circuit C_2 has $n_2 = 2^r k$ input variables and size $m_2 \leq 2m_1 + 2^{2r+1}k$, such that C_1 is satisfiable if and only if (C_2, k) is a yes-instance of WCS*$^*[t]$*. The running time of the algorithm A_1 is bounded* *by $O(m_2^2)$.*

Proof. Let $k = \lceil n_1/r \rceil$. Divide the n_1 input variables $x_1, \ldots, x_{n_1}$ of the Π_t-circuit C_1 into k blocks $B_1, \ldots, B_k$, where block B_i consists of input variables $x_{(i-1)r+1}, \ldots, x_{ir}$, for $i = 1, \ldots, k-1$, and block B_k consists of input variables $x_{(k-1)r+1}, \ldots, x_{n_1}$. Denote by $|B_i|$ the number of variables in block B_i. Then $|B_i| = r$, for $1 \le i \le k-1$, and $|B_k| \le r$. For an integer j, $0 \le j \le 2^{|B_i|} - 1$, denote by $\mathrm{bin}_i(j)$ the length-$|B_i|$ binary representation of j, which can also be interpreted as an assignment to the variables in block B_i.

We construct a new set of input variables in k blocks $B'_1, \ldots, B'_k$. Each block B'_i consists of $s = 2^r$ variables $z_{i,0}$, $z_{i,1}$, $\ldots$, $z_{i,s-1}$. The Π_t-circuit C_2 is constructed from the Π_t-circuit C_1 by replacing the input gates in C_1 by the new input variables in $B'_1, \ldots, B'_k$. We consider two cases.

Case 1. t is even. Then all level-1 gates in the Π_t-circuit C_1 are OR gates. We connect the new variables $z_{i,j}$ to these level-1 gates to construct the circuit C_2 as follows. Let x_q be an input variable in C_1 such that x_q is the h-th variable in block B_i. If the positive literal x_q is an input to a level-1 OR gate g_1 in C_1, then all positive literals $z_{i,j}$ in block B'_i such that $0 \le j \le 2^{|B_i|} - 1$ and the h-th bit in $\mathrm{bin}_i(j)$ is 1 are connected to gate g_1 in the circuit C_2. If the negative literal $\overline{x}_q$ is an input to a level-1 OR gate g_2 in C_1, then all positive literals $z_{i,j}$ in block B'_i such that $0 \le j \le 2^{|B_i|} - 1$ and the h-th bit in $\mathrm{bin}_i(j)$ is 0 are connected to gate g_2 in the circuit C_2.

Note that if the size $|B_k|$ of the last block B_k in C_1 is smaller than r, then the above construction for block B'_k is only on the first $2^{|B_k|}$ variables in B'_k, and the last $s - 2^{|B_k|}$ variables in B'_k have no output edges, and hence become "dummy variables".

We also add an "enforcement" circuitry to the circuit C_2 to ensure that every satisfying assignment to C_2 assigns the value 1 to at least one variable in each block

B'_i. This can be achieved by having an OR gate for each block B'_i, whose inputs are connected to all positive literals in block B'_i and whose output is an input to the output gate of the circuit C_2 (for block B'_k, the inputs of the OR gate are from the first $2^{|B_k|}$ variables in B'_k). This completes the construction of the circuit C_2. It is easy to see that the circuit C_2 is a monotone Π_t-circuit (note that $t \geq 2$ and hence the enforcement circuitry does not increase the depth of C_2). Thus, (C_2, k) is an instance of the problem WCS$^+[t]$.

We verify that the circuit C_1 is satisfiable if and only if the circuit C_2 has a satisfying assignment of weight k. Suppose that the circuit C_1 is satisfied by an assignment τ. Let τ_i be the restriction of τ to block B_i, $1 \leq i \leq k$. Let j_i be the integer such that $\text{bin}_i(j_i) = \tau_i$. Then according to the construction of the circuit C_2, by setting $z_{i,j_i} = 1$ and all other variables in B'_i to 0, we can satisfy all level-1 OR gates in C_2 whose corresponding level-1 OR gates in C_1 are satisfied by the assignment τ_i. Doing this for all blocks B_i, $1 \leq i \leq k$, gives a weight-k assignment τ' to the circuit C_2 that satisfies all level-1 OR gates in C_2 whose corresponding level-1 OR gates in C_1 are satisfied by τ. Since τ satisfies the circuit C_1, the weight-k assignment τ' satisfies the circuit C_2.

Conversely, suppose that the circuit C_2 is satisfied by a weight-k assignment τ'. Because of the enforcement circuitry in C_2, τ' assigns the value 1 to exactly one variable in each block B'_i (in particular, in block B'_k, this variable must be one of the first $2^{|B_k|}$ variables in B'_k). Now suppose that in block B'_i, τ' assigns the value 1 to the variable z_{i,j_i}. Then we set an assignment τ_i to the block B_i in C_1 such that $\tau_i = \text{bin}_i(j_i)$. By the construction of the circuit C_2, the level-1 OR gates satisfied by the variable $z_{i,j_i} = 1$ are all satisfied by the assignment τ_i. Therefore, if we make an assignment τ to the circuit C_1 such that the restriction of τ to block B_i is τ_i for all i, then the assignment τ will satisfy all level-1 OR gates in C_1 whose corresponding level-1 OR gates in C_2 are satisfied by τ'. Since τ' satisfies the circuit C_2, we conclude that the circuit C_1 is satisfiable.

This completes the proof that when t is even, the circuit C_1 is satisfiable if and only if the constructed pair (C_2, k) is a yes-instance of WCS$^+[t]$.

Case 2. t is odd. Then all level-1 gates in the Π_t-circuit C_1 are AND gates. We connect the new variables $z_{i,j}$ to these level-1 gates to construct the circuit C_2 as follows. Let x_q be an input variable in C_1 and be the h-th variable in block B_i. If the positive literal x_q is an input to a level-1 AND gate g_1 in C_1, then all negative literals $\overline{z}_{i,j}$ in block B'_i such that $0 \leq j \leq 2^{|B_i|} - 1$ and the h-th bit in bin$_i(j)$ is 0 are inputs to gate g_1 in C_2. If the negative literal $\overline{x}_q$ is an input to a level-1 AND gate g_2 in C_1, then all negative literals $\overline{z}_{i,j}$ in block B'_i such that $0 \leq j \leq 2^{|B_i|} - 1$ and the h-th bit in bin$_i(j)$ is 1 are inputs to gate g_2 in C_2.

For the last $s - 2^{|B_k|}$ variables in the last block B'_k in C_2, we connect the negative literals $\overline{z}_{k,j}$, $2^{|B_k|} \leq j \leq s-1$, to the output gate of the circuit C_2 (thus, the variables $z_{k,j}$, $2^{|B_k|} \leq j \leq s-1$, are forced to have the value 0 in any satisfying assignment to C_2).

An enforcement circuitry is added to C_2 to ensure that every satisfying assignment to C_2 assigns the value 1 to at most one variable in each block B'_i. This can be achieved as follows. For every two distinct negative literals $\overline{z}_{i,j}$ and $\overline{z}_{i,h}$ in B'_i, $0 \leq j, h \leq 2^{|B_i|} - 1$, add an OR gate $g_{j,h}$. Connect $\overline{z}_{i,j}$ and $\overline{z}_{i,h}$ to $g_{i,h}$ and connect $g_{i,h}$ to the output AND gate of C_2. This completes the construction of the circuit C_2. The circuit C_2 is an antimonotone Π_t-circuit (again the enforcement circuitry does not increase the depth of C_2). Thus, (C_2, k) is an instance of the problem WCS$^-[t]$.

We verify that the circuit C_1 is satisfiable if and only if the circuit C_2 has a satisfying assignment of weight k. Suppose that the circuit C_1 is satisfied by an assignment τ. Let τ_i be the restriction of τ to block B_i, $1 \leq i \leq k$. Let j_i be the integer such that $\mathrm{bin}_i(j_i) = \tau_i$. Consider the weight-$k$ assignment τ' to C_2 that for each i assigns $z_{i,j_i} = 1$ and all other variables in B'_i to 0. We show that τ' satisfies the circuit C_2. Let g_1 be a level-1 AND gate in C_1 that is satisfied by the assignment τ. Since C_2 is antimonotone, all inputs to g_1 in C_2 are negative literals. Since all negative literals except $\overline{z}_{i,j_i}$ in block B'_i have the value 1, we only have to prove that no $\overline{z}_{i,j_i}$ from any block B'_i is an input to g_1. Assume to the contrary that $\overline{z}_{i,j_i}$ in block B'_i is an input to g_1. Then by the construction of the circuit C_2, there is a variable x_q that is the h-th variable in block B_i such that either x_q is an input to g_1 in C_1 and the h-th bit of $\mathrm{bin}_i(j_i)$ is 0, or $\overline{x}_q$ is an input to g_1 in C_1 and the h-th bit of $\mathrm{bin}_i(j_i)$ is 1. However, by our construction of the index j_i from the assignment τ, if the h-th bit of $\mathrm{bin}_i(j_i)$ is 0 then τ assigns $x_q = 0$, and if the h-th bit of $\mathrm{bin}_i(j_i)$ is 1 then τ assigns $x_q = 1$. In either case, τ would not satisfy the gate g_1, contradicting our assumption. Thus, for all i, no $\overline{z}_{i,j_i}$ is an input to the gate g_1, and the assignment τ' satisfies the gate g_1. Since g_1 is an arbitrary level-1 AND gate in C_2, we conclude that the assignment τ' satisfies all level-1 AND gates in C_2 whose corresponding gates in C_1 are satisfied by the assignment τ. Since τ satisfies the circuit C_1, the weight-k assignment τ' satisfies the circuit C_2.

Conversely, suppose that the circuit C_2 is satisfied by a weight-k assignment τ'. Because of the enforcement circuitry in C_2, the assignment τ' assigns the value 1 to exactly one variable in each block B'_i (in particular, this variable in block B'_k must be one of the first $2^{|B_k|}$ variables in B'_k since the last $s - 2^{|B_k|}$ variables in B'_k are forced to have the value 0 in the satisfying assignment τ'). Suppose that in block B'_i, τ' assigns the value 1 to the variable z_{i,j_i}. Then we set an assignment $\tau_i = \mathrm{bin}_i(j_i)$ to block B_i in C_1. Let τ be the assignment whose restriction on block B_i is τ_i. We prove that τ satisfies the circuit C_1. In effect, if a level-1 AND gate g_2 in C_2 is satisfied by the assignment τ', then no negative literal $\overline{z}_{i,j_i}$ is an input to g_2. Suppose that g_2 is not satisfied by τ in C_1, then either a positive literal x_q is an input to g_2 and τ assigns $x_q = 0$, or a negative literal $\overline{x}_q$ is an input to g_2 and τ assigns $x_q = 1$. Let x_q be the h-th variable in block B_i. If τ assigns $x_q = 0$ then the h-th bit in $\text{bin}_i(j_i)$ is 0. Thus, x_q cannot be an input to g_2 in C_1 because otherwise by our construction the negative literal $\overline{z}_{i,j_i}$ would be an input to g_2 in C_2. On the other hand, if τ assigns $x_q = 1$ then the h-th bit in $\text{bin}_i(j_i)$ is 1, thus, $\overline{x}_q$ cannot be an input to g_2 in C_1 because otherwise the negative literal $\overline{z}_{i,j_i}$ would be an input to g_2 in C_2. This contradiction shows that the gate g_2 must be satisfied by the assignment τ. Since g_2 is an arbitrary level-1 AND gate in C_2, we conclude that the assignment τ satisfies all level-1 AND gates in C_1 whose corresponding level-1 AND gates in C_2 are satisfied by the assignment τ'. Since τ' satisfies the circuit C_2, the assignment τ satisfies the circuit C_1 and hence the circuit C_1 is satisfiable.

This completes the proof that when t is odd, the Π_t-circuit C_1 is satisfiable if and only if the pair (C_2, k) is a yes-instance of WCS$^-[t]$.

Summarizing the above discussion, we conclude that for any $t \geq 2$, from a Π_t-circuit C_1 of n_1 input variables and size m_1, we can construct an instance (C_2, k) of the problem WCS$^*[t]$ such that C_1 is satisfiable if and only if (C_2, k) is a yes-instance of WCS$^*[t]$. Here $k = \lceil n_1/r \rceil$, and C_2 has $n_2 = 2^r k$ input variables and size $m_2 \leq m_1 + n_2 + k + k2^{2r} \leq 2m_1 + k2^{2r+1}$ (where the term $k + k2^{2r}$ is an upper bound on the size of the enforcement circuitry). Finally, it is straightforward to verify that the pair (C_2, k) can be constructed from the circuit C_1 in time $O(m_2^2)$. $\square$

Lemma V.1 will serve as a basis for proving computational lower bounds for $W[2]$-hard problems. In order to derive similar computational lower bounds for certain $W[1]$-hard problems, we need another lemma that converts weighted satisfiability problems on monotone CNF formulas into weighted satisfiability problems on anti-monotone CNF formulas.

The parameterized problem Weighted Monotone CNF 2-SAT, abbreviated WCNF 2-SAT$^+$ (resp. Weighted Antimonotone CNF 2-SAT, abbreviated WCNF 2-SAT$^-$) is: given an integer k and a CNF formula F, in which all literals are positive (resp. negative) and each clause contains at most 2 literals, determine whether there is a satisfying assignment of weight k to F.

Lemma V.2 *There is an algorithm A_2 that, for a given integer $r > 0$, transforms each instance (F_1, k_1) of WCNF 2-SAT$^+$, where the formula F_1 has n_1 variables, into a group $\mathcal{G}$ of at most $(r+1)^{k_2}$ instances (F_π, k_2) of WCNF 2-SAT$^-$, where $k_2 = \lceil n_1/r \rceil$, and each formula F_π has $n_2 = k_2 2^r$ variables, such that (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$ if and only if there is a yes-instance for WCNF 2-SAT$^-$ in the group $\mathcal{G}$. The running time of the algorithm A_2 is bounded by $O(n_2^2 (r+1)^{k_2})$.*

Proof. For the given instance (F_1, k_1) of WCNF 2-SAT$^+$, divide the n_1 variables in F_1 into $k_2 = \lceil n_1/r \rceil$ pairwise disjoint subsets B_1, …, B_{k_2}, each containing at most r variables. Let π be a partition of the parameter k_1 into k_2 integers h_1, …, h_{k_2}, where $0 \le h_i \le |B_i|$ and $k_1 = h_1 + \cdots, h_{k_2}$. We say that an assignment τ of weight k_1 for F_1 is *under the partition* π if τ assigns the value 1 to exactly h_i variables in the set B_i for every i.

Fix a partition π of the parameter k_1: $k_1 = h_1 + \cdots + h_{k_2}$. We construct an instance (F_π, k_2) for WCNF 2-SAT$^-$ as follows. For each subset $B_{i,j}$ of h_i variables in the set B_i, if for each clause (x_s, x_t) in F_1 where both x_s and x_t are in B_i, at least one of x_s and x_t is in $B_{i,j}$, then make $B_{i,j}$ a Boolean variable in F_π. Call such a $B_{i,j}$ an "essential variable" in F_π. In particular, if no clause (x_s, x_t) in F_1 has both x_s and x_t in the set B_i, then every subset of h_i variables in B_i makes an essential variable in F_π. For each pair of essential variables $B_{i,j}$ and $B_{i,q}$ in F_π from the same set B_i in F_1, add a clause $(\overline{B_{i,j}}, \overline{B_{i,q}})$ to F_π. For each pair of essential variables $B_{i,j}$ and $B_{h,q}$

in F_π from two different sets B_i and B_h in F_1, if there exist a variable $x_s \in B_i$ and a variable $x_t \in B_h$ such that $x_s \notin B_{i,j}$, $x_t \notin B_{h,q}$ but (x_s, x_t) is a clause in F_1, add a clause $(\overline{B_{i,j}}, \overline{B_{h,q}})$ to F_π. This completes the main part of the CNF formula F_π, which thus far has no more than $k_2 2^r$ variables. To make the number n_2 of variables in F_π to be exactly $k_2 2^r$, we add a proper number of "surplus" variables to F_π and for each surplus variable B' we add a unit clause $(\overline{B'})$ to F_π (so that these surplus variables are forced to have the value 0 in a satisfying assignment of F_π). Obviously, (F_π, k_2) is an instance of the WCNF 2-SAT$^-$ problem.

We verify that the CNF formula F_1 has a satisfying assignment of weight k_1 under the partition π if and only if the CNF formula F_π has a satisfying assignment of weight k_2. Let τ_1 be a satisfying assignment of weight k_1 under the partition π for F_1. Let C be the set of variables in F_1 that are assigned the value 1 by τ_1, and $C_i = C \cap B_i$. Then C_i has h_i variables. Note that for any clause (x_s, x_t) in F_1 such that both x_s and x_t are in B_i, at least one of x_s and x_t must be in C_i – otherwise the clause (x_s, x_t) would not be satisfied by the assignment τ_1. Thus, each subset C_i is an essential variable in F_π. Now in the CNF formula F_π, by assigning the value 1 to all C_i, $1 \le i \le k_2$, and the value 0 to all other variables (in particular, all surplus variables in F_π are assigned the value 0), we get an assignment τ_π of weight k_2 for F_π. For each clause of the form $(\overline{B_{i,j}}, \overline{B_{i,q}})$ in F_π, where $B_{i,j}$ and $B_{i,q}$ are from the same set B_i, since only one variable in F_π from the set B_i (i.e., C_i) is assigned the value 1 by τ_π, the clause is satisfied by the assignment τ_π. For two variables C_i and C_h in F_π, $i \neq h$, which both get assigned the value 1 by the assignment τ_π, each clause (x_s, x_t) in F_1 such that $x_s \in B_i$ and $x_t \in B_h$ must have either $x_s \in C_i$ or $x_t \in C_h$ (otherwise the clause (x_s, x_t) would not be satisfied by τ_1). Thus, $(\overline{C_i}, \overline{C_h})$ is not a clause in F_π. In consequence, the clauses of the form $(\overline{B_{i,j}}, \overline{B_{h,q}})$ in F_π, $i \neq h$, where $B_{i,j}$ and $B_{h,q}$ are from different sets B_i and B_h, are also all satisfied by τ_π. This shows that F_π is satisfied by the assignment τ_π of weight k_2.

Conversely, let τ_π be a satisfying assignment of weight k_2 for F_π. Because $(\overline{B_{i,j}}, \overline{B_{i,q}})$ is a clause in F_π for each pair of essential variables $B_{i,j}$ and $B_{i,q}$ from the same set B_i, at most one essential variable in F_π from each set B_i can be assigned the value 1 by the assignment τ_π. Since the weight of τ_π is k_2, we conclude that exactly one essential variable B_{i,j_i} in F_π from each set B_i is assigned the value 1 by τ_π (note that all surplus variables in F_π must be assigned the value 0 by τ_π). Each B_{i,j_i} of these subsets in F_1 contains exactly h_i variables in B_i. Let $C = \cup_{i=1}^{k_2} B_{i,j_i}$, then C has exactly k_1 variables in F_1. If in F_1 we assign all variables in C the value 1 and all other variables the value 0, we get an assignment τ_1 of weight k_1 for the formula F_1. We show that τ_1 is a satisfying assignment for F_1. For each clause (x_s, x_t) in F_1 where both x_s and x_t are in the same set B_i, by the construction of the essential variables in F_π, at least one of x_s and x_t is in B_{i,j_i}, and hence in C. Thus, all clauses (x_s, x_t) in F_1 where both x_s and x_t are in B_i are satisfied by the assignment τ_1. For each clause (x_s, x_t) in F_1 where $x_s \in B_i$ and $x_t \in B_h$, $i \neq h$, because $(\overline{B_{i,j_i}}, \overline{B_{h,j_h}})$ is not a clause in F_π (otherwise, τ_π would not satisfy F_π), we must have either $x_s \in B_{i,j_i}$ or $x_t \in B_{h,j_h}$, i.e., at least one of x_s and x_t must be in C. It follows that the clause (x_s, x_t) is again satisfied by τ_1. This proves that τ_1 is a satisfying assignment of weight k_1 for the formula F_1.

For each partition π of the parameter k_1, we have a corresponding instance (F_π, k_2) such that the CNF formula F_1 has a satisfying assignment of weight k_1 under the partition π if and only if (F_π, k_2) is a yes-instance of WCNF 2-SAT$^-$. Let $\mathcal{G}$ be the collection of the instances (F_π, k_2) over all partitions π of the parameter k_1. Since (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$ if and only if there is a partition π of k_1 such that F_1 has a satisfying assignment of weight k_1 under the partition π, we conclude that (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$ if and only if the group $\mathcal{G}$ contains a yes-instance of WCNF 2-SAT$^-$. The number of instances in the group $\mathcal{G}$ is bounded by the number of partitions of k_1, which is bounded by $(r+1)^{k_2}$. Finally, the instance (F_π, k_2) for a partition π of k_1 can be constructed in time $O(n_2^2)$. Therefore, the group $\mathcal{G}$ of the instances of WCNF 2-SAT$^-$ can be constructed in time $O(n_2^2(r+1)^{k_2})$. This completes the proof of the lemma. $\square$

C. Lower Bounds on Weighted Satisfiability Problems

From Lemma V.1, we can get a number of interesting results on the relationship between the circuit satisfiability problem SAT$[t]$ and the weighted circuit satisfiability problem WCS$^*[t]$.

In the following theorems, we will denote by n the number of input variables and m the size of a circuit.

Theorem V.3 *Let $t \geq 2$ be an integer. For any function f, if the problem WCS$^*[t]$ is solvable in time $f(k)n^{o(k)}m^{O(1)}$, then the problem SAT[t] can be solved in time $2^{o(n)}m^{O(1)}$.*

PROOF. Suppose that there is an algorithm M_{WCS} of running time bounded by $f(k)n^{k/\lambda(k)}p(m)$ that solves the problem WCS$^*[t]$, where $\lambda(k)$ is a nondecreasing and unbounded function and p is a polynomial. Without loss of generality, we can assume that the function f is nondecreasing, unbounded, and that $f(k) \geq 2^k$. Define f^{-1} by $f^{-1}(h) = \max\{q \mid f(q) \leq h\}$. Since the function f is nondecreasing and unbounded, the function f^{-1} is also nondecreasing and unbounded, and satisfies $f(f^{-1}(h)) \leq h$. From $f(k) \geq 2^k$, we have $f^{-1}(h) \leq \log h$.

Now we solve the problem SAT$[t]$ as follows. For an instance C_1 of SAT$[t]$, where C_1 is a Π_t-circuit of n_1 input variables and size m_1, we set the integer $r =$

$\lfloor 3n_1/f^{-1}(n_1)\rfloor$, and call the algorithm A_1 in Lemma V.1 to convert C_1 into an instance (C_2, k) of the problem WCS*$[t]$. Here $k = \lceil n_1/r\rceil$, C_2 is a Π_t-circuit of $n_2 = 2^r k$ input variables and size $m_2 \le 2m_1 + 2^{2r+1}k$, and the algorithm A_1 takes time $O(m_2^2)$. According to Lemma V.1, we can determine if C_1 is a yes-instance of SAT$[t]$ by calling the algorithm $M_{\rm WCS}$ to determine if (C_2, k) is a yes-instance of WCS*$[t]$. The running time of the algorithm $M_{\rm WCS}$ on (C_2, k) is bounded by $f(k)n_2^{k/\lambda(k)}p(m_2)$. Combining all above we get an algorithm $M_{\rm sat}$ of running time $f(k)n_2^{k/\lambda(k)}p(m_2) + O(m_2^2)$ for the problem SAT$[t]$. We analyze the running time of the algorithm $M_{\rm sat}$ in terms of the values n_1 and m_1.

Since $k = \lceil n_1/r\rceil \le f^{-1}(n_1) \le \log n_1$,[2] we have $f(k) \le f(f^{-1}(n_1)) \le n_1$. Moreover,

$$k = \lceil n_1/r\rceil \ge n_1/r \ge n_1/(3n_1/f^{-1}(n_1)) = f^{-1}(n_1)/3$$

Therefore if we set $\lambda'(n_1) = \lambda(f^{-1}(n_1)/3)$, then $\lambda(k) \ge \lambda'(n_1)$. Since both λ and f^{-1} are nondecreasing and unbounded, $\lambda'(n_1)$ is a nondecreasing and unbounded function of n_1. We have (note that $k \le f^{-1}(n_1) \le \log n_1$),

$$n_2^{k/\lambda(k)} = (k2^r)^{k/\lambda(k)} \le k^k 2^{kr/\lambda(k)} \le k^k 2^{3kn_1/(\lambda(k)f^{-1}(n_1))} \le k^k 2^{3n_1/\lambda(k)} \le k^k 2^{3n_1/\lambda'(n_1)} = 2^{o(n_1)}$$

Finally, consider the factor m_2. Since f^{-1} is nondecreasing and unbounded,

$$m_2 \le 2m_1 + k2^{2r+1} \le 2m_1 + 2\log n_1 2^{6n_1/f^{-1}(n_1)} = 2^{o(n_1)}m_1$$

Therefore, both terms $p(m_2)$ and $O(m_2^2)$ in the running time of the algorithm $M_{\rm sat}$ are bounded by $2^{o(n_1)}p'(m_1)$ for a polynomial p'. Combining all these, we conclude that the running time $f(k)n_2^{k/\lambda(k)}p(m_2) + O(m_2^2)$ of $M_{\rm sat}$ is bounded by $2^{o(n_1)}p'(m_1)$

[2]Without loss of generality, we assume that in our discussions, all values under the ceiling function "$\lceil\cdot\rceil$" and the floor function "$\lfloor\cdot\rfloor$" are greater than or equal to 1. Therefore, we will always assume the inequalities $\lceil\beta\rceil \le 2\beta$ and $\lfloor\beta\rfloor \ge \beta/2$ for any value β.

for a polynomial p'. Hence, the problem SAT[t] can be solved in time $2^{o(n)}m^{O(1)}$. This completes the proof of the theorem. □

In fact, Theorem V.3 remains valid even if we restrict the parameter values to be bounded by an arbitrarily small function, as shown in the following corollary.

Corollary V.4 *Let $t \geq 2$ be an integer, and $\mu(n)$ a nondecreasing and unbounded function. If for a function f, the problem WCS***[t] is solvable in time $f(k)n^{o(k)}m^{O(1)}$ for parameter values $k \leq \mu(n)$, then the problem SAT[t] can be solved in time $2^{o(n)}m^{O(1)}$.*

Proof. Suppose that there is an algorithm M solving the WCS*[t] problem in time $f(k)n^{o(k)}p(m)$ for parameter values $k \leq \mu(n)$, where p is a polynomial. Define $\mu^{-1}(h) = \max\{q \mid \mu(q) \leq h\}$. Since the function μ is nondecreasing and unbounded, the function μ^{-1} is also nondecreasing, unbounded, and such that $k > \mu(n)$ implies $n \leq \mu^{-1}(k)$.

Now we develop an algorithm that solves the WCS*[t] problem for general parameter values. For a given instance (C, k) of WCS*[t], if $k > \mu(n)$ then we enumerate all weight-k assignments to the circuit C and check if any of them satisfies the circuit, and if $k \leq \mu(n)$, we call the algorithm M to decide if (C, k) is a yes-instance for WCS*[t]. This algorithm obviously solves the problem WCS*[t]. Moreover, in case $k > \mu(n)$, the algorithm runs in time $O(2^n m^2) = O(f_1(k)m^2)$, where $f_1(k) = 2^{\mu^{-1}(k)}$, while in case $k \leq \mu(n)$, the algorithm runs in time $f(k)n^{o(k)}p(m)$. Therefore, the algorithm solves the problem WCS*[t] for general parameter values in time $O(f_2(k)n^{o(k)}m^{O(1)})$, where $f_2(k) = \max\{f(k), f_1(k)\}$. Now the corollary follows from Theorem V.3. □

Further extension of the above techniques shows that, essentially, Theorem V.3 remains true for *every* parameter value.

Theorem V.5 *Let $t \geq 2$ be an integer and ϵ be a fixed constant, $0 < \epsilon < 1$. For any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^{\epsilon}$ and $\mu(2n) \leq 2\mu(n)$, if WCS*$[t]$ is solvable in time $n^{o(k)}m^{O(1)}$ for parameter values $\mu(n)/8 \leq k \leq 16\mu(n)$, then SAT$[t]$ is solvable in time $2^{o(n)}m^{O(1)}$.*

PROOF. We first show that by properly choosing the number r in Lemma V.1, we can make the parameter value $k = \lceil n_1/r \rceil$ satisfy the condition $\mu(n_2)/8 \leq k \leq 16\mu(n_2)$, where $n_2 = k2^r$. To show this, we extend the function μ to a continuous function by connecting $\mu(i)$ and $\mu(i+1)$ by a linear function for each integer i.

Fix the value n_1, and consider the function

$$F(z) = \mu\left(\frac{n_1 2^{z \log n_1}}{z \log n_1}\right) - \frac{n_1}{z \log n_1} = \mu\left(\frac{n_1^{z+1}}{z \log n_1}\right) - \frac{n_1}{z \log n_1}$$

Pick a real number z_0, $0 < z_0 < 1$, such that $(z_0 \log n_1)^{1-\epsilon} \leq n_1^{1-(z_0+1)\epsilon}$. For this value z_0, since $\mu(n_1^{z_0+1}/(z_0 \log n_1)) \leq (n_1^{z_0+1}/(z_0 \log n_1))^{\epsilon} \leq n_1/(z_0 \log n_1)$, we have $F(z_0) \leq 0$. Moreover, it is easy to check that $F(n_1/\log n_1) \geq 0$. Therefore, there is a real number z^* between z_0 and $n_1/\log n_1$ such that

$$\mu\left(\frac{n_1 2^{z^* \log n_1}}{z^* \log n_1}\right) \leq \frac{n_1}{z^* \log n_1} \qquad \text{and} \qquad \mu\left(\frac{n_1 2^{z^* \log n_1 + 1}}{z^* \log n_1 + 1}\right) \geq \frac{n_1}{z^* \log n_1 + 1} \quad (5.1)$$

We explain how to find such a real number z^* efficiently. Starting from the value z_0, then the integer values $z_1 = 1$, $z_2 = 2$, ..., $\lceil n_1/\log n_1 \rceil$, we find the smallest z_i such that

$$\mu\left(\frac{n_1 2^{z_i \log n_1}}{z_i \log n_1}\right) \leq \frac{n_1}{z_i \log n_1} \qquad \text{and} \qquad \mu\left(\frac{n_1 2^{z_{i+1} \log n_1}}{z_{i+1} \log n_1}\right) \geq \frac{n_1}{z_{i+1} \log n_1}$$

Now check the values $z_{i,j} = z_i + j/\log n_1$ for $j = 0, 1, \ldots, \lceil \log n_1 \rceil$ to find a j such that

$$\mu\left(\frac{n_1 2^{z_{i,j}\log n_1}}{z_{i,j}\log n_1}\right) \le \frac{n_1}{z_{i,j}\log n_1} \qquad \text{and} \qquad \mu\left(\frac{n_1 2^{z_{i,j+1}\log n_1}}{z_{i,j+1}\log n_1}\right) \ge \frac{n_1}{z_{i,j+1}\log n_1}$$

Note that $z_{i,j+1} = z_{i,j} + 1/\log n_1$ so $z_{i,j+1}\log n_1 = z_{i,j}\log n_1 + 1$. Thus, we can set $z^* = z_{i,j}$.

Now we have

$$2\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) \ge 2\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1 + 1}\right) \ge \mu\left(\frac{n_1 2^{z^*\log n_1 + 1}}{z^*\log n_1 + 1}\right) \ge \frac{n_1}{z^*\log n_1 + 1} \ge \frac{n_1}{2z^*\log n_1} \tag{5.2}$$

where the second inequality uses the fact $2\mu(n) \ge \mu(2n)$. From (5.1) and (5.2), we get

$$4\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) \ge \frac{n_1}{z^*\log n_1} \ge \mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) \tag{5.3}$$

Therefore, if we set $r = \lceil z^*\log n_1\rceil$, then from $k = \lceil n_1/r\rceil$, $n_2 = 2^r k$, and (5.3), we have

$$\begin{aligned}
\mu(n_2) &= \mu(2^r k) = \mu(2^r\lceil n_1/r\rceil) \ge \mu(2^r n_1/r) \ge \mu\left(\frac{2^{z^*\log n_1} n_1}{2z^*\log n_1}\right) \\
&\ge \frac{1}{2}\mu\left(\frac{2^{z^*\log n_1} n_1}{z^*\log n_1}\right) \ge \frac{1}{8}\cdot\frac{n_1}{z^*\log n_1} \ge \frac{1}{8}\cdot\frac{n_1}{\lceil z^*\log n_1\rceil} = \frac{1}{8}\cdot\frac{n_1}{r} \ge \frac{1}{16}\cdot\lceil n_1/r\rceil = \frac{k}{16}
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\mu(n_2) &= \mu(2^r k) \le \mu(2^{z^*\log n_1 + 1} k) \le 2\mu(2^{z^*\log n_1}\lceil n_1/r\rceil) \le 2\mu(2^{z^*\log n_1 + 1} n_1/r) \\
&\le 4\mu\left(\frac{2^{z^*\log n_1} n_1}{z^*\log n_1}\right) \le \frac{4n_1}{z^*\log n_1} \le \frac{8n_1}{\lceil z^*\log n_1\rceil} = \frac{8n_1}{r} \le 8\lceil n_1/r\rceil = 8k
\end{aligned}$$

This proves that the values k and n_2 satisfy the relation $\mu(n_2)/8 \le k \le 16\mu(n_2)$.

Now we are ready to prove our theorem. Suppose that there is an algorithm M_{WCS} of running time $n^{k/\lambda(k)}p(m)$ for the WCS*$[t]$ problem when the parameter values k are in the range $\mu(n)/8 \le k \le 16\mu(n)$, where $\lambda(k)$ is a nondecreasing and unbounded function and p is a polynomial. We solve the problem SAT$[t]$ as follows:

For an instance C_1 of SAT$[t]$, where C_1 is a Π_t-circuit of n_1 input variables and size m_1,

(A) Let $r = \lceil z^* \log n_1 \rceil$, where z^* is the real number satisfying (5.1). As we explained above, the value z^* can be computed in time polynomial in n_1;

(B) Call the algorithm A_1 in Lemma V.1 on r and C_1 to construct an instance (C_2, k) of the problem WCS$^*[t]$, where $k = \lceil n_1/r \rceil$, and C_2 is a Π_t-circuit of $n_2 = k2^r$ input variables and size $m_2 \leq 2m_1 + 2^{2r+1}k$. By the above discussion, we have $\mu(n_2)/8 \leq k \leq 16\mu(n_2)$;

(C) Call the algorithm M_{WCS} on (C_2, k) to determine whether (C_2, k) is a yes-instance of WCS$^*[t]$, which, by Lemma V.1, is equivalent to whether C_1 is a yes-instance of SAT$[t]$.

The running time of steps (A) and (B) of the above algorithm is bounded by a polynomial $p_1(m_2)$ of m_2. Step (C) takes time $n_2^{k/\lambda(k)} p(m_2)$. Therefore, the total running time of this algorithm solving the SAT$[t]$ problem is bounded by $n_2^{k/\lambda(k)} p_2(m_2)$, where p_2 is a polynomial. We have (for simplicity and without affecting the correctness, we omit the floor and ceiling functions),

$$n_2^{k/\lambda(k)} = (2^r n_1/r)^{(n_1/r)/\lambda(n_1/r)} \leq 2^{n_1/\lambda(n_1/r)} n_1^{(n_1/r)/\lambda(n_1/r)}$$

Now it is easy to verify that $n_2^{k/\lambda(k)} = 2^{o(n_1)}$ (observe that $k = n_1/r \geq \mu(n_2)/8$ hence $\lambda(n_1/r)$ is unbounded, and that $r = z^* \log n_1 = \Omega(\log n_1)$). Also, since $m_2 \leq 2m_1 + 2(n_2)^2$, $m_2 = 2^{o(n_1)} m_1^{O(1)}$, thus, the polynomial $p_2(m_2)$ is bounded by $2^{o(n_1)} m_1^{O(1)}$. This concludes that the above algorithm of running time $n_2^{k/\lambda(k)} p_2(m_2)$ for the problem SAT$[t]$ has its running time bounded by $2^{o(n_1)} m_1^{O(1)}$. This completes the proof of the theorem. □

Now we derive similar results for the weighted satisfiability problem WCNF 2-SAT$^-$, based on Lemma V.2. In the following discussion, for an instance (F, k) of the problems WCNF 2-SAT$^-$ or WCNF 2-SAT$^+$, we denote by n and m, respectively, the number of variables and the instance size of the CNF formula F. Note that $m = O(n^2)$.

Theorem V.6 *If the problem WCNF 2-SAT$^-$ is solvable in time $f(k)m^{o(k)}$ for a function f, then the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$.*

Proof. Since $m = O(n^2)$ for any instance of WCNF 2-SAT$^-$, we only need to prove that if the problem WCNF 2-SAT$^-$ is solvable in time $f(k)n^{o(k)}$ for a function f, then the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$.

Suppose that the problem WCNF 2-SAT$^-$ is solvable in time $f(k)n^{k/\lambda(k)}$ for a nondecreasing and unbounded function λ. Without loss of generality, we can assume that the function f is nondecreasing, unbounded, and satisfies $f(k) > 2^k$. Define $f^{-1}(h) = \max\{q \mid f(q) \leq h\}$. Then f^{-1} is a nondecreasing and unbounded function satisfying $f^{-1}(h) \leq \log h$ and $f(f^{-1}(h)) \leq h$.

For a given instance (F_1, k_1) of WCNF 2-SAT$^+$, where the CNF formula F_1 has n_1 variables, we let $r = \lfloor 3n_1/f^{-1}(n_1) \rfloor$ and $k_2 = \lceil n_1/r \rceil$, then we use the algorithm A_2 in Lemma V.2 to construct a group $\mathcal{G}$ of at most $(r+1)^{k_2}$ instances (F_π, k_2) of WCNF 2-SAT$^-$, where each formula F_π has $n_2 = k_2 2^r$ variables, and such that (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$ if and only if the group $\mathcal{G}$ contains a yes-instance of WCNF 2-SAT$^-$. By our assumption, it takes time $f(k_2)n_2^{k_2/\lambda(k_2)}$ to test if each (F_π, k_2) in the group $\mathcal{G}$ is a yes-instance of WCNF 2-SAT$^-$. Therefore, in time of order

$$(r+1)^{k_2} f(k_2) n_2^{k_2/\lambda(k_2)} + n_2^2 (r+1)^{k_2}$$

we can decide if (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$, where the term $n_2^2(r+1)^{k_2}$ is for the running time of the algorithm A_2. As we verified in Theorem V.3, $f(k_2) \leq n_1$, and $n_2^{k_2/\lambda(k_2)} = 2^{o(n_1)}$ (in particular, $n_2 = 2^{o(n_1)}$). Finally, since $r = O(n_1)$ and $k_2 = O(f^{-1}(n_1)) = O(\log n_1)$, we get $(r+1)^{k_2} = 2^{o(n_1)}$. In summary, in time $2^{o(n_1)}$ we can decide if (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$, and hence, the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$. □

Based on Theorem V.6, and using a proof completely similar to that of Corollary V.4, we can prove that Theorem V.6 remains valid even if we restrict the parameter values to be bounded by an arbitrarily small function of n.

Corollary V.7 *Let $\mu(n)$ be any nondecreasing and unbounded function. If there is a function f such that the problem WCNF 2-SAT$^-$ is solvable in time $f(k)m^{o(k)}$ for parameter values $k \leq \mu(n)$, then the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$.*

Theorem V.8 *For any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^\epsilon$ and $\mu(2n) \leq 2\mu(n)$, where ϵ is a fixed constant, $0 < \epsilon < 1$, if WCNF 2-SAT$^-$ is solvable in time $m^{o(k)}$ for parameter values $\mu(n)/8 \leq k \leq 16\mu(n)$, then the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$.*

Proof. Again since $m = O(n^2)$, the given hypothesis implies that WCNF 2-SAT$^-$ is solvable in time $n^{o(k)}$ for parameter values $\mu(n)/8 \leq k \leq 16\mu(n)$.

Let (F_1, k_1) be an instance of WCNF 2-SAT$^+$, where the CNF formula F_1 has n_1 variables. As in Theorem V.5, we first compute in polynomial time a real number z^* satisfying

$$4\mu\left(\frac{n_1 2^{z^* \log n_1}}{z^* \log n_1}\right) \geq \frac{n_1}{z^* \log n_1} \geq \mu\left(\frac{n_1 2^{z^* \log n_1}}{z^* \log n_1}\right)$$

Now we let $r = \lceil z^* \log n_1 \rceil$ and $k_2 = \lceil n_1/r \rceil$, and use the algorithm A_2 in Lemma V.2 to construct a group $\mathcal{G}$ of at most $(r+1)^{k_2}$ instances (F_π, k_2) of WCNF 2-SAT$^-$, where each formula F_π has $n_2 = k_2 2^r$ variables, such that (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$ if and only if the group $\mathcal{G}$ contains a yes-instance of WCNF 2-SAT$^-$.

As proved in Theorem V.5, the values k_2 and n_2 satisfy the relation $\mu(n_2)/8 \leq k_2 \leq 16\mu(n_2)$, and $n_2^{k_2/\lambda(k_2)} = 2^{o(n_1)}$ for any nondecreasing and unbounded function λ. Therefore, by the hypothesis of the current theorem, we can determine in time $2^{o(n_1)}$ for each (F_π, k_2) in $\mathcal{G}$ if (F_π, k_2) is a yes-instance of WCNF 2-SAT$^-$. It is also easy to verify that the total number $(r+1)^{k_2}$ of instances in the group $\mathcal{G}$ and the running time $O(n_2^2(r+1)^{k_2})$ of the algorithm A_2 are all bounded by $2^{o(n_1)}$. Therefore, using this transformation, we can determine in time $2^{o(n_1)}$ whether (F_1, k_1) is a yes-instance of WCNF 2-SAT$^+$, and hence the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n_1)}$.

$\square$

D. Satisfiability Problems and the W-hierarchy

We first show that a subexponential time algorithm for SAT[t] would collapse the W-hierarchy.

Theorem V.9 *For any integer $t \geq 2$, if SAT[t] is solvable in time $2^{o(n)}m^{O(1)}$, then $W[t-1] = FPT$.*

Proof. The theorem for the case $t = 2$ is an easy corollary of Corollary 3.1 in [14]. Here we present a proof for the general case $t \geq 3$ using different techniques. In particular, our techniques do not apply to the case $t = 2$.

Let C be a Π_{t-1}-circuit of n input variables $x_0, \ldots, x_{n-1}$ and size m such that C is monotone if t is odd and C is antimonotone if t is even. Without loss of generality, we assume that $\log n$ is an integer (otherwise, we add dummy input variables to C). Let $k \leq n$ be a non-negative integer. We first show how to construct a Π_t-circuit

C' of $k \log n$ input variables from the circuit C and the integer k such that C has a satisfying assignment of weight k if and only if C' is satisfiable. The input variables in C' are divided into k blocks $B'_1, \ldots, B'_k$, where each block B'_i consists of $r = \log n$ input variables $z_{i,1}, \ldots, z_{i,r}$. For a non-negative integer $j \leq n-1$, we denote by $\text{bin}_r(j)$ the length-r binary representation of the integer j, which can also be interpreted as an assignment to a block B'_i in the circuit C'. We distinguish two cases based on the parity of t.

Case 1. t is odd. Then C is a monotone Π_{t-1}-circuit and all level-1 gates in C are OR gates. For each positive literal x_j in C and for each block B'_i, we associate an AND gate $g_{i,j}$ in C' such that if the h-th bit in $\text{bin}_r(j)$ is 1 (resp. 0) then $z_{i,h}$ (resp. $\overline{z}_{i,h}$) is an input to $g_{i,j}$. The outputs of $g_{i,j}$ in C' are identical to the outputs of x_j in C. Note that for each assignment $\text{bin}_r(j)$ to block B'_i, exactly one of these new AND gates, i.e., the gate $g_{i,j}$, is satisfied and outputs 1. Thus, the assignment $\text{bin}_r(j)$ of block B'_i in C' simulates the assignment $x_j = 1$ in C. The circuit C' is obtained from the circuit C by removing all input gates in C and adding the kn new AND gates $g_{i,j}$, $1 \leq i \leq k$, $0 \leq j \leq n-1$, and the literals in blocks B'_1, …, B'_k. Moreover, we add an enforcement circuitry to C' to make sure that the assignments to different blocks in C' simulate assignments to different variables in C. To achieve this, we construct a depth-2 subcircuit $C_{i,i'}$ for each pair of blocks B'_i and $B'_{i'}$ such that $C_{i,i'}$ outputs 0 if and only if blocks B'_i and $B'_{i'}$ are assigned the same value. The output of $C_{i,i'}$ is an input to the output AND gate of the circuit C'. Since $t \geq 3$, the enforcement circuitry does not increase the depth of the circuit C'. Thus, the circuit C' is a Π_t-circuit with kr input variables.

It is easy to verify that the circuit C has a satisfying assignment of weight k if and only if the circuit C' is satisfiable: suppose C is satisfied by a weight-k assignment τ, which assigns the value 1 to k variables x_{j_1}, …, x_{j_k}, and the value 0 to all other variables. Then by assigning the value $\text{bin}_r(j_i)$ to block B'_i for $1 \leq i \leq k$, we get an assignment τ' for the circuit C' such that all AND gates g_{i,j_i} in C' are satisfied. Since the outputs of the AND gates g_{i,j_i} are identical to the outputs of the positive literals x_{j_i}, we conclude that all level-2 OR gates in C' corresponding to those level-1 OR gates in C satisfied by the assignment τ are satisfied by the assignment τ'. Since the assignment τ satisfies the circuit C and all blocks B'_i are assigned different values, the assignment τ' satisfies the circuit C' and the circuit C' is satisfiable. Conversely, suppose the circuit C' is satisfied by an assignment τ', then the restriction τ'_i of τ' to block B'_i satisfies exactly one AND gate g_{i,j_i}, where $\text{bin}_r(j_i) = \tau'_i$. Because of the enforcement circuitry, these k gates g_{i,j_i} correspond to k different positive literals x_{j_i}. Thus, if we set $x_{j_i} = 1$ for all $1 \leq i \leq k$, and assign the value 0 to all other variables, we get an assignment τ of weight exactly k that satisfies the circuit C.

Case 2. t is even. Then C is an antimonotone Π_{t-1}-circuit and all level-1 gates in C are AND gates. For each input variable x_j, $0 \leq j \leq n-1$, and for each block B'_i, we make an OR gate $g_{i,j}$ such that if the h-th bit in $\text{bin}_r(j)$ is 0 (resp. 1) then $z_{i,h}$ (resp. $\overline{z}_{i,h}$) is an input to $g_{i,j}$. The outputs of $g_{i,j}$ in C' are identical to the outputs of $\overline{x}_j$ in C. Note that for each assignment $\text{bin}_r(j)$ of block B'_i, exactly one of these new OR gates, i.e., the gate $g_{i,j}$, is not satisfied and outputs 0. Thus, the assignment $\text{bin}_r(j)$ of block B'_i in C' simulates the assignment $\overline{x}_j = 0$ (or equivalently $x_j = 1$) in C. As in Case 1, we also add an enforcement circuitry to C' to make sure that no two blocks in C' are assigned the same value. The circuit C' is a Π_t-circuit with kr input variables.

To verify that the circuit C has a satisfying assignment of weight k if and only if the circuit C' is satisfiable, suppose C is satisfied by a weight-k assignment τ, which assigns the value 1 to k variables $x_{j_1}, \ldots, x_{j_k}$, and the value 0 to all other variables. Then by assigning the value $\text{bin}_r(j_i)$ to block B'_i for $1 \leq i \leq k$, we get an assignment

τ' to the circuit C' such that for each i, only the OR gate g_{i,j_i} is not satisfied and outputs 0. Thus, for each level-1 AND gate g_1 satisfied by the assignment τ in C, since no negative literals $\overline{x}_{j_1}$, …, $\overline{x}_{j_k}$ are inputs to g_1 in C, no gates g_{1,j_1}, …, g_{k,j_k} are inputs to g_1 in C'. Thus, the assignment τ' satisfies the gate g_1. Since g_1 is an arbitrary level-1 AND gate satisfied by τ in C, we conclude that the assignment τ' satisfies all level-2 AND gates that correspond to the level-1 AND gates satisfied by the assignment τ in C. Since τ satisfies the circuit C and all blocks B'_i are assigned different values, τ' satisfies the circuit C' and C' is satisfiable. Conversely, suppose the circuit C' is satisfied by an assignment τ', then the restriction τ'_i of τ' to block B'_i satisfies all OR gates $g_{i,j}$ except the gate g_{i,j_i}, where $\mathrm{bin}_r(j_i) = \tau'_i$. Because of the enforcement circuitry in C', assignments τ'_i and $\tau'_{i'}$ to two different blocks in C' are different. Thus, the assignments to the k blocks induce k different input variables x_{j_i}. If we set $x_{j_i} = 1$ for all $1 \le i \le k$ and set the value 0 for all other input variables in C, we get an assignment τ of weight exactly k satisfying the circuit C.

In summary, we have verified that for any $t \ge 3$, for a given Π_{t-1}-circuit C of n input variables and size m, and for a given $k \le n$, where C is monotone if t is odd and antimonotone if t is even, we can construct a Π_t-circuit C' such that C has a satisfying assignment of weight k if and only if C' is satisfiable. The circuit C' has $n' = kr = k \log n$ input variables and size m' bounded by $m + kn + 3k^2 \log^2 n \le 3m^3$, where the term kn is the number of the gates $g_{i,j}$, $1 \le i \le k$, $0 \le j \le n-1$ in the construction of the circuit C', and $3k^2 \log^2 n$ is an upper bound on the size of the enforcement circuitry. The circuit C' can be constructed from (C, k) in time $O((m')^2)$.

By the hypothesis of the theorem, there is an algorithm A' that determines whether the circuit C' is satisfiable in time $2^{o(n')}p(m')$ for a polynomial p. Thus, there is a nondecreasing and unbounded function λ such that the running time of the algorithm A' is bounded by $2^{n'/\lambda(n')}p(m')$. This, plus the construction of the circuit C' from (C, k), gives an algorithm A'' of running time $2^{n'/\lambda(n')}p_1(m')$ that determines whether the Π_{t-1}-circuit C has a satisfying assignment of weight k, where p_1 is a polynomial. Note that $2^{n'/\lambda(n')} = 2^{k \log n/\lambda(k \log n)} \leq 2^{k \log n/\lambda(\log n)}$. This gives the following algorithm A that solves the WCS*$[t-1]$ problem:

> For a given instance (C, k) of WCS*$[t-1]$, where C has n input variables and size m, if $k > \lambda(\log n)$, then enumerate all assignments to C and check if there is a satisfying assignment of weight k to C; if $k \leq \lambda(\log n)$, then call the algorithm A'' to decide if there is a satisfying assignment of weight k to C.

We analyze the algorithm A. First note that $m' \leq 3m^3$, thus, $p_1(m')$ is bounded by a polynomial $p'(m)$ of m. Define $\lambda^{-1}(h) = \min\{q \mid \lambda(q) \geq h\}$. Since λ is nondecreasing and unbounded, λ^{-1} is also a nondecreasing and unbounded function. Let $f(k) = 2^{2^{\lambda^{-1}(k)}}$. We claim that the running time of the algorithm A is bounded by $f(k)np'(m)$. In effect, if $k > \lambda(\log n)$, we have $\lambda^{-1}(k) \geq \log n$, and $f(k) \geq 2^n$. Therefore, in this case, the running time of the algorithm A is bounded by $2^n p'(m) \leq f(k)p'(m)$. On the other hand, if $k \leq \lambda(\log n)$, then the algorithm A calls the algorithm A'' to solve the problem, which runs in time $2^{k \log n/\lambda(\log n)} \leq 2^{\log n} = n$.

Thus, under the hypothesis of the theorem, we have been able to prove that the $W[t-1]$-complete problem WCS*$[t-1]$ is solvable in time $f(k)np'(m)$ for a function f and a polynomial p', and hence is fixed-parameter tractable. This, in consequence, implies that $W[t-1] = FPT$. □

Combining Theorem V.9 with Theorem V.3, Corollary V.4, and Theorem V.5, we get

Theorem V.10 *For any integer $t \geq 2$, if the problem WCS*$[t]$ is solvable in time $f(k)n^{o(k)}m^{O(1)}$ for a function f, then $W[t-1] = FPT$. This theorem remains true* *even if we restrict the parameter values k by $k \leq \mu(n)$ for any nondecreasing and unbounded function μ.*

Theorem V.11 *Let $t \geq 2$ be an integer and ϵ be a fixed constant, $0 < \epsilon < 1$. For any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^{\epsilon}$ and $\mu(2n) \leq 2\mu(n)$, if the problem WCS*[t] is solvable in time $n^{o(k)}m^{O(1)}$ for the parameter values $\mu(n)/8 \leq k \leq 16\mu(n)$, then $W[t-1] = FPT$.*

Now we consider the satisfiability problems WCNF 2-SAT$^-$ and WCNF 2-SAT$^+$on CNF formulas. In the following discussion, for an instance (F, k) of the problems WCNF 2-SAT$^-$ or WCNF 2-SAT$^+$, we denote by n and m, respectively, the number of variables and the instance size of the formula F. Note that $m = O(n^2)$.

The class SNP introduced by Papadimitriou and Yannakakis [67] contains many well-known NP-hard problems including, for any fixed integer $q \geq 3$, CNF q-SAT, q-Colorability, q-Set Cover, and Vertex Cover, Clique, and Independent Set [15]. It is commonly believed that it is unlikely that all problems in SNP are solvable in subexponential time[3]. Impagliazzo and Paturi [15] studied the class SNP and identified a group of SNP-complete problems under the SERF-reduction, in the sense that if any of these SNP-complete problems is solvable in subexponential time, then all problems in SNP are solvable in subexponential time.

Lemma V.12 *If the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$, then all problems in SNP are solvable in subexponential time.*

PROOF. It is easy to see that the problem Vertex Cover can be reduced to the

[3]A recent result showed the equivalence between the statement that all SNP problems are solvable in subexponential time, and the collapse of a parameterized class called *Mini*[1] to *FPT* [82].

problem WCNF 2-SAT$^+$ in a straightforward way: given an instance (G, k) of Vertex Cover, where G is a graph of n vertices, we can construct an instance (F_G, k) of WCNF 2-SAT$^+$, where the CNF formula F_G has n variables, as follows: each vertex v_i of G makes a positive literal x_i in F_G, and each edge $[v_i, v_j]$ in G makes a clause (x_i, x_j) in F_G. It is easy to see that the graph G has a vertex cover of k vertices if and only if the CNF formula F_G has a satisfying assignment of weight k. Therefore, if the problem WCNF 2-SAT$^+$ is solvable in time $2^{o(n)}$, then the problem Vertex Cover is solvable in subexponential time. Since Vertex Cover is SNP-complete under the SERF-reduction [15], this in consequence implies that all problems in SNP are solvable in subexponential time. $\square$

Combining Lemma V.12 with Theorem V.6, Corollary V.7, and Theorem V.8, we get

Theorem V.13 *If the problem WCNF 2-SAT$^-$ is solvable in time $f(k)m^{o(k)}$ for a function f, then all problems in SNP are solvable in subexponential time. This theorem remains true even if we restrict the parameter values k by $k \leq \mu(n)$ for any nondecreasing and unbounded function μ.*

Theorem V.14 *For any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^\epsilon$ and $\mu(2n) \leq 2\mu(n)$, where ϵ is a fixed constant, $0 < \epsilon < 1$, if WCNF 2-SAT$^-$ is solvable in time $m^{o(k)}$ for parameter values $\mu(n)/8 \leq k \leq 16\mu(n)$, then all problems in SNP are solvable in subexponential time.*

E. Linear fpt-reductions and Lower Bounds

In the discussion of the problems WCS$^*[t]$, we observed that besides the parameter k and the circuit size m, the number n of input variables has played an important role in the computational complexity of the problems. Unless unlikely collapses occur in parameterized complexity theory, the problems WCS$^*[t]$ require computational time $f(k)n^{\Omega(k)}p(m)$, for any polynomial p and any function f. The dominating term in the time bound depends on the number n of input variables in the circuits, instead of the circuit size m. Note that the circuit size m can be of the order 2^n.

Each instance (C, k) of a weighted circuit satisfiability problem such as WCS$^*[t]$ can be regarded as a search problem, in which we need to select k elements from a search space consisting of a set of n input variables, and assign them the value 1 so that the circuit C is satisfied. Many well-known NP-hard problems have similar formulations. We list some of them next:

> Weighted CNF SAT (abbreviated WCNF-SAT): given a CNF formula F, and an integer k, decide if there is an assignment of weight k that satisfies all clauses in F. Here the search space is the set of Boolean variables in F.
>
> Set Cover: given a collection $\mathcal{F}$ of subsets in a universal set U, and an integer k, decide whether there is a subcollection of k subsets in $\mathcal{F}$ whose union is equal to U. Here the search space is $\mathcal{F}$.
>
> Hitting Set: given a collection $\mathcal{F}$ of subsets in a universal set U, and an integer k, decide if there is a subset S of k elements in U such that S intersects every subset in $\mathcal{F}$. Here the search space is U.

Many graph problems seek a subset of vertices that meet certain given conditions. For these graph problems, the natural search space is the set of all vertices. For certain problems, a polynomial time preprocessing on the input instance can significantly reduce the size of the search space. For example, for finding a vertex cover of k vertices in a graph G of n vertices, a polynomial time preprocessing can reduce the search space size to $2k$ (see Proposition II.1). In the following, we present a simple algorithm for reducing the search space size for the Dominating Set problem.

Suppose we are looking for a dominating set of k vertices in a graph G. Without loss of generality, we assume that G contains no isolated vertices (otherwise, we simply include the isolated vertices in the dominating set and modify the graph G and the parameter k accordingly). We say that the graph G has an *IS-Clique partition* (V_1, V_2) if the vertices of G can be partitioned into two disjoint subsets V_1 and V_2 such that V_1 makes an independent set while V_2 induces a clique. If $|V_2| \leq k$, then the vertices in V_2 plus any $k - |V_2|$ vertices in V_1 make a dominating set of k vertices in G. Thus, we assume that $|V_2| > k$. We claim that the graph G has a dominating set of k vertices if and only if there are k vertices in V_2 that make a dominating set for G. In fact, suppose that G has a dominating set D of k vertices, in which k_1 are in V_1 and k_2 are in V_2, where $k_1 + k_2 = k$. Now for each vertex v in $D \cap V_1$ that has no neighbor in D, we replace in D the vertex v by a neighbor u of v such that u is in V_2 (such a neighbor u must exist since V_1 is an independent set and v is not an isolated vertex). This process gives us a dominating set D' of at most k vertices in G, where D' is a subset of V_2. Adding a proper number of vertices in V_2 to D' then gives a dominating set of exact k vertices in G.

Therefore, if we are looking for a dominating set of k vertices in a graph G with an IS-Clique partition (V_1, V_2), we can restrict our search to the set of vertices in V_2, which thus makes a search space for the problem. Now we explain how to test if a given graph G has an IS-Clique partition.

Lemma V.15 *Let the vertices of G be ordered as $\{v_1, v_2, \ldots, v_n\}$ such that $deg(v_1) \leq deg(v_2) \leq \cdots \leq deg(v_n)$ (where $deg(v_i)$ denotes the degree of the vertex v_i). If $G = (V, E)$ has an IS-Clique partition, then either there is a vertex v_i in G where v_i and its neighbors make a clique V_2 such that $(V - V_2, V_2)$ makes an IS-Clique partition* *for G, or there is an index h, $1 \leq h \leq n-1$, such that deg$(v_h) <$ deg(v_{h+1}) and $(\{v_1, \ldots, v_h\}, \{v_{h+1}, \ldots, v_n\})$ is an IS-Clique partition for G.*

Proof. Suppose that the graph G has an IS-Clique partition (V_1, V_2). We consider three different cases. (1) If there is a vertex v_i in V_2 such that v_i has no neighbor in V_1, then v_i and its neighbors make exactly the set V_2 and (V_1, V_2) is an IS-Clique partition for G; (2) If there is a vertex v_j in V_1 that is adjacent to all vertices in V_2, then v_j and its neighbors make the set $V_2 \cup \{v_j\}$, and $(V_1 - \{v_j\}, V_2 \cup \{v_j\})$ is an IS-Clique partition for G; (3) If neither of (1) and (2) is the case, then each vertex in V_2 has degree at least $|V_2|$ and each vertex in V_1 has degree at most $|V_2| - 1$. □

Using Lemma V.15, we can develop a simple algorithm of running time $O(n^3)$ that tests if a given graph has an IS-Clique partition. Summarizing the above we obtain the following preprocessing algorithm on an instance (G, k) of the Dominating Set problem:

```
DS-Core(G, k)
1. if the graph G has no IS-Clique partition, then let U be the entire set of
   vertices in G;
2. else construct an IS-Clique partition (V_1, V_2) for G;
        if |V_2| < k, Then let U be V_2 plus any k − |V_2| vertices in V_1;
        else let U = V_2;
3. return U as the search space.
```

Fig. 11. The algorithm DS-Core

The parameterized problems discussed in the current chapter all share the property that they seek a subset in a search space satisfying certain properties. In most of the problems that we consider, the search space can be easily identified. For example, the search space for each of the problems WCNF-SAT, Set Cover, and Hitting Set is given as we described. For some other problems, such as Dominating Set, the search space can be identified by a polynomial time preprocessing algorithm (such as the **DS-Core** algorithm Figure 11). If no polynomial time preprocessing algorithm is known, then we simply pick the entire input instance as the search space. For example, for the problems Independent Set and Clique, we will take the search space to be the entire vertex set. Thus, each instance of our parameterized problems is associated with a triple (k, n, m), where k is the parameter, n is the size of the search space, and m is the size of the instance. We will call such an instance a (k, n, m)-*instance.*

Theorems V.10 and V.13 suggest that the problem WCS$^*[t]$ in the class $W[t]$ for $t \geq 2$ and the problem WCNF 2-SAT$^-$ in the class $W[1]$ seem to have very high parameterized complexity. In the following, we introduce a new reduction to identify problems in the corresponding classes that are at least as difficult as these problems.

Definition V.16 *A parameterized problem Q is linearly fpt-reducible (shortly fpt$_l$-reducible) to a parameterized problem Q' if there exist a function f and an algorithm A of running time $f(k)n^{o(k)}m^{O(1)}$, such that on each (k, n, m)-instance x of Q, the algorithm A produces a (k', n', m')-instance x' of Q', where $k' = O(k)$, $n' = n^{O(1)}$, $m' = m^{O(1)}$, and that x is a yes-instance of Q if and only if x' is a yes-instance of Q'.*

Definition V.17 *A parameterized problem Q_1 is $W[1]$-hard under the linear fpt-reduction, shortly $W_l[1]$-hard, if the problem WCNF 2-SAT$^-$ is fpt$_l$-reducible to Q_1. A parameterized problem Q_t is $W[t]$-hard under the linear fpt-reduction, shortly $W_l[t]$-hard, for $t \geq 2$ if the problem WCS$^*[t]$ is fpt$_l$-reducible to Q_t.*

Based on the above definitions and using Theorem V.10 and Theorem V.13, we immediately derive:

Theorem V.18 *For $t \geq 2$, no $W_l[t]$-hard parameterized problem can be solved in time $f(k)n^{o(k)}m^{O(1)}$ for a function f, unless $W[t-1] = FPT$. This remains true* *even if we restrict the parameter values k by $k \leq \mu(n)$ for any nondecreasing and unbounded function μ.*

Theorem V.19 *No $W_l[1]$-hard parameterized problem can be solved in time $f(k)m^{o(k)}$ for a function f, unless all problems in SNP are solvable in subexponential time. This remains true even if we restrict the parameter values k by $k \leq \mu(n)$ for any nondecreasing and unbounded function μ.*

Using the fpt$_l$-reduction, we can immediately derive computational lower bounds for a large number of NP-hard parameterized problems.

Theorem V.20 *The following parameterized problems are $W_l[2]$-hard: WCNF-SAT, Set Cover, Hitting Set, and Dominating Set. Thus, unless $W[1] = FPT$, none of them can be solved in time $f(k)n^{o(k)}m^{O(1)}$ for any function f. This theorem remains true even if we restrict the parameter values k by $k \leq \mu(n)$ for any nondecreasing and unbounded function μ.*

Proof. We highlight the fpt$_l$-reductions from WCS$^*[2]$ = WCS$^+[2]$ to these problems, which are all we need. In fact, the reductions from WCS$^+[2]$ to the problems WCNF-SAT, Hitting Set, and Set Cover are standard and straightforward, and hence we leave them to the interested readers.

We present the fpt$_l$-reduction from WCS$^+[2]$ to Dominating Set here. Let (C, k) be an instance of WCS$^+[2]$, where C is a monotone Π_2-circuit. We construct a graph G_C associated with the circuit C as follows. First we remove any OR gate in C if it receives inputs from all input gates (this kind of OR gates will be satisfied by any assignment of weight larger than 0 anyway). Then we remove the output gate of C and add an edge to each pair of input gates in C. This gives the graph G_C. We claim that the circuit C has a satisfying assignment of weight k if and only if the graph G_C has a dominating set of k vertices. First observe that the graph G_C has a unique IS-Clique partition (V_1, V_2), where V_1 is the set of all OR gates and V_2 is the set of all input gates. Therefore, by the discussion before Lemma V.15, if G_C has a dominating set D of k vertices, then we can assume that D is a subset of V_2. Now assigning the value 1 to the k input variables corresponding to the vertices in D clearly gives a satisfying assignment of weight k for the circuit C. For the other direction, from a satisfying assignment π of weight k for the circuit C, we can easily verify that the k vertices in G_C corresponding to the k input gates in C assigned the value 1 by π make a dominating set for the graph G_C. Finally, we point out that this reduction keeps the parameter value k, the search space size n (assuming that we apply the algorithm **DS-Core** to the Dominating Set problem), and the instance size m all unchanged. $\square$

We remark that the reduction from WCS$^+$[2] to Dominating Set presented in the proof of Theorem V.20 also provides a new proof for the $W[2]$-hardness for the problem Dominating Set, which seems to be significantly simpler than the original proof given in [11].

Now we consider certain $W_l[1]$-hard problems. Define WCNF q-SAT, where $q > 0$ is a fixed integer, to be the parameterized problem consisting of the pairs (F, k), where F is a CNF formula in which each clause contains at most q literals and F has a satisfying assignment of weight k.

Theorem V.21 *The following problems are $W_l[1]$-hard: WCNF q-SAT for any integer $q \geq 2$, Clique, and Independent Set. Thus, unless all problems in SNP are solvable in subexponential time, none of them can be solved in time $f(k)m^{o(k)}$ for any function f. This theorem remains true even if we restrict the parameter values k by $k \leq \mu(m)$ for any nondecreasing and unbounded function μ.*

PROOF. The fpt$_l$-reductions from the problem WCNF 2SAT$^-$ to these problems are all straightforward, and hence we leave the detailed verifications to the interested readers. □

Each of the problems in Theorem V.20 and Theorem V.21 can be solved by a trivial algorithm of running time $cn^k m^2$, where c is an absolute constant, which simply enumerates all possible subsets of k elements in the search space. Much research has tended to seek new approaches to improve this trivial upper bound. One of the common approaches is to apply a more careful branch-and-bound search process trying to optimize the manipulation of local structures before each branch [73, 74, 12, 75, 76]. Continuously improved algorithms for these problems have been developed based on improved local structure manipulations. It has even been proposed to automate the manipulation of local structures [47, 77] in order to further improve the computational time.

Theorem V.20 and Theorem V.21, however, provide strong evidence that the power of this approach is quite limited in principle. The lower bound $f(k)n^{\Omega(k)}p(m)$ for the problems in Theorem V.20 and the lower bound $f(k)m^{\Omega(k)}$ for the problems in Theorem V.21, where f can be any function and p can be any polynomial, indicate that *no* local structure manipulation running in polynomial time or in time depending only on the target value k will obviate the need for exhaustive enumerations.

Weaker lower bounds, under the same assumptions in parameterized complexity theory, have been established previously [60] for the parameterized problems in Theorem V.20 and Theorem V.21. The main results in [60] proved that, for the case $k = \sqrt{n/\log n}$, an algorithm of running time $n^{o(k)}m^{O(1)}$ for the problems in Theorem V.20 would imply $W[1] = FPT$, and an algorithm of running time $m^{o(k)}$ for the problems in Theorem V.21 would imply that all problems in SNP are subexponential time solvable. However, the results in [60] do not exclude the possibility of algorithms of running time $f(k)n^{o(k)}m^{O(1)}$ for the problems in Theorem V.20, and algorithms of running time $f(k)m^{o(k)}$ for the problems in Theorem V.21, where f can be possibly a very large function. Moreover, the results in [60] do not claim lower bounds for the problems when the parameter value k is not equal to $\sqrt{n/\log n}$. Note that studying the complexity of NP-hard problems for parameter values other than $\sqrt{n/\log n}$, in particular for small parameter values, has been an interesting topic in research [83, 84]. Moreover, after all, most research in parameterized complexity theory assumes that the parameter values are small. Therefore, Theorem V.20 and Theorem V.21 are very significant improvements over the results in [60].

One might suspect that a particular parameter value (e.g., a very small parameter value or a very large parameter value) would help solving the problems in Theorem V.20 and Theorem V.21 more efficiently. This possibility is, unfortunately, denied by the following theorems, which indicate that, essentially, the problems are actually difficult for *every* parameter value.

Theorem V.22 *For any constant ϵ, $0 < \epsilon < 1$, and any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^{\epsilon}$, and $\mu(2n) \leq 2\mu(n)$, none of the problems in Theorem V.20 can be solved in time $n^{o(k)}m^{O(1)}$ even if we restrict the parameter values k to $\mu(n)/8 \leq k \leq 16\mu(n)$, unless $W[1] = FPT$.*

PROOF. As described in the proof of Theorem V.20, each fpt$_l$-reduction from WCS$^+[2]$ to a problem in Theorem V.20 runs in time $m^{O(1)}$ and keeps the parameter value k and the search space size n unchanged. The theorem now follows directly from this fact and Theorem V.11. $\Box$

Note that the conditions on the function μ in Theorem V.22 are satisfied by most complexity functions, such as $\mu(n) = \log\log n$ and $\mu(n) = n^{4/5}$. Therefore, for example, unless the unlikely collapse $W[1] = FPT$ occurs, constructing a dominating set of $\log\log n$ vertices requires time $n^{\Omega(\log\log n)}m^{O(1)}$, and constructing a dominating set of $\sqrt{n}$ vertices requires time $n^{\Omega(\sqrt{n})}m^{O(1)}$.

Similar results hold for the problems in Theorem V.21, by similar proofs based on Theorem V.14.

Theorem V.23 *For any constant ϵ, $0 < \epsilon < 1$, and any nondecreasing and unbounded function μ satisfying $\mu(n) \leq n^{\epsilon}$, and $\mu(2n) \leq 2\mu(n)$, none of the problems in Theorem V.21 can be solved in time $m^{o(k)}$ even if we restrict the parameter values k to $\mu(n)/8 \leq k \leq 16\mu(n)$, unless all problems in SNP are subexponential time solvable.*

We observe that all problems in Theorem V.20 are also $W_l[1]$-hard. Thus, we can actually claim stronger lower bounds for these problems in terms of the parameter value k and the instance size m, based on a stronger assumption [4]. This result will be used in the next section.

Theorem V.24 *All problems in Theorem V.20 are $W_l[1]$-hard. Hence, none of them can be solved in time $f(k)m^{o(k)}$ for any function f, unless all SNP problems are subexponential time solvable.*

Proof. The fpt$_l$-reduction from WCNF 2-SAT$^-$ to WCNF-SAT is straightforward. It is not difficult to verify that the fpt-reduction from WCNF-SAT to Dominating Set described in [11], which was originally used to prove the $W[2]$-hardness for Dominating Set, is actually an fpt$_l$-reduction. Finally, the fpt$_l$-reduction from Dominating Set to Hitting Set, and the fpt$_l$-reduction from Hitting Set to Set Cover are simple and

[4]It can be shown that if $W[1] = FPT$ then all problems in SNP are solvable in subexponential time.

left to the interested readers. The theorem now follows from the transitivity of the fpt$_l$-reduction, which can be easily verified. □

F. Lower Bounds on Approximation Schemes

In this section, we discuss how the $W_l[1]$-hardness of a problem can be used to derive computational lower bounds for approximation algorithms for NP-hard problems. We first give a brief review on the terminologies in approximation algorithms.

An *NP optimization problem* Q is a 4-tuple (I_Q, S_Q, f_Q, opt_Q), where

1. I_Q is the set of input instances. It is recognizable in polynomial time;
2. For each instance $x \in I_Q$, $S_Q(x)$ is the set of feasible solutions for x, which is defined by a polynomial p and a polynomial time computable predicate π (p and π only depend on Q) as $S_Q(x) = \{y : |y| \le p(|x|) \text{ and } \pi(x, y)\}$;
3. $f_Q(x, y)$ is the objective function mapping a pair $x \in I_Q$ and $y \in S_Q(x)$ to a non-negative integer. The function f_Q is computable in polynomial time;
4. $opt_Q \in \{\max, \min\}$. Q is called a *maximization problem* if $opt_Q = \max$, and a *minimization problem* if $opt_Q = \min$.

An *optimal solution* y_0 for an instance $x \in I_Q$ is a feasible solution in $S_Q(x)$ such that $f_Q(x, y_0) = opt_Q\{f_Q(x, z) \mid z \in S_Q(x)\}$. We will denote by $opt_Q(x)$ the value $opt_Q\{f_Q(x, z) \mid z \in S_Q(x)\}$.

An algorithm A is an *approximation algorithm* for an NP optimization problem Q if, for each input instance x in I_Q, the algorithm A returns a feasible solution $y_A(x)$ in $S_Q(x)$. The solution $y_A(x)$ has an *approximation ratio* $r(n)$ if it satisfies the following condition:

$$opt_Q(x)/f_Q(x, y_A(x)) \leq r(|x|) \quad \text{if } Q \text{ is a maximization problem}$$

$$f_Q(x, y_A(x))/opt_Q(x) \leq r(|x|) \quad \text{if } Q \text{ is a minimization problem}$$

The approximation algorithm A has an *approximation ratio* $r(m)$ if for any instance x in I_Q, the solution $y_A(x)$ constructed by the algorithm A has an approximation ratio bounded by $r(|x|)$.

An NP optimization problem Q has a *polynomial time approximation scheme* (PTAS) if there is an algorithm A_Q that takes a pair (x, ϵ) as input, where x is an instance of Q and $\epsilon > 0$ is a real number, and returns a feasible solution y for x such that the approximation ratio of the solution y is bounded by $1 + \epsilon$, and for each fixed $\epsilon > 0$, the running time of the algorithm A_Q is bounded by a polynomial of $|x|$.

We propose the following formal framework for parameterization of NP optimization problems.

Definition V.25 *Let $Q = (I_Q, S_Q, f_Q, opt_Q)$ be an NP optimization problem. The parameterized version of Q is defined as follows:*

1. *If Q is a maximization problem, then the parameterized version of Q is defined as $Q_{\geq} = \{(x, k) \mid x \in I_Q \text{ and } opt_Q(x) \geq k\}$;*
2. *If Q is a minimization problem, then the parameterized version of Q is defined as $Q_{\leq} = \{(x, k) \mid x \in I_Q \text{ and } opt_Q(x) \leq k\}$.*

The above definition offers the possibility to study the relationship between the approximability and the parameterized complexity of NP optimization problems.

Theorem V.26 *Let Q be an NP optimization problem. If the parameterized version of Q is $W_l[1]$-hard, then Q has no PTAS of running time $f(1/\epsilon)m^{o(1/\epsilon)}$ for any function f, unless all problems in SNP are solvable in subexponential time.*

Proof. We consider the case that $Q = (I_Q, S_Q, f_Q, opt_Q)$ is a maximization problem such that the parameterized version $Q_\geq$ of Q is $W_l[1]$-hard.

Suppose to the contrary that Q has a PTAS A_Q of running time $f(1/\epsilon)m^{o(1/\epsilon)}$ for a function f. We show how to use the algorithm A_Q to solve the parameterized problem $Q_\geq$. Consider the following algorithm $A_\geq$ for $Q_\geq$:

> Algorithm $A_\geq$:
>
> On an instance (x, k) of $Q_\geq$, call the PTAS algorithm A_Q on x and $\epsilon = 1/(2k)$. Suppose that A_Q returns a solution y in $S_Q(x)$. If $f_Q(x, y) \geq k$, then return "yes", otherwise return "no".

We verify that the algorithm $A_\geq$ solves the parameterized problem $Q_\geq$. Since Q is a maximization problem, if $f_Q(x, y) \geq k$ then obviously $opt_Q(x) \geq k$. Thus, the algorithm $A_\geq$ returns a correct decision in this case. On the other hand, suppose $f_Q(x, y) < k$. Since $f_Q(x, y)$ is an integer, we have $f_Q(x, y) \leq k - 1$. Since A_Q is a PTAS for Q and $\epsilon = 1/(2k)$, we must have

$$opt_Q(x)/f_Q(x, y) \leq 1 + 1/(2k)$$

From this we get (note that $f_Q(x, y) < k$)

$$opt_Q(x) \leq f_Q(x, y) + f_Q(x, y)/(2k) \leq k - 1 + 1/2 = k - 1/2 < k$$

Thus, in this case the algorithm $A_\geq$ also returns a correct decision. This proves that the algorithm $A_\geq$ solves the parameterized version $Q_\geq$ of the problem Q. The running time of the algorithm $A_\geq$ is dominated by that of the algorithm A_Q, which by our hypothesis is bounded by $f(1/\epsilon)m^{o(1/\epsilon)} = f(2k)m^{o(k)}$. Thus, the $W_l[1]$-hard problem $Q_\geq$ is solvable in time $f(2k)m^{o(k)}$. By Theorem V.19, all problems in SNP are solvable in subexponential time.

The proof is similar for the case when Q is a minimization problem, and hence is omitted. □

We demonstrate an application for Theorem V.26. We pick the NP-complete problem Distinguishing Substring Selection as an example, which has drawn a lot of attention recently because of its applications in computational biology such as in drug generic design [79].

Consider all strings over a fixed alphabet. Denote by $|s|$ the length of the string s. The *distance* $D(s_1, s_2)$ between two strings s_1 and s_2, $|s_1| \leq |s_2|$, is defined as follows. If $|s_1| = |s_2|$, then $D(s_1, s_2)$ is the Hamming distance between s_1 and s_2, and if $|s_1| \leq |s_2|$, then $D(s_1, s_2)$ is the minimum of $D(s_1, s_2')$ over all substrings s_2' of length $|s_1|$ in s_2.

> Distinguishing Substring Selection (DSSP): given a tuple (n, S_b, S_g, d_b, d_g), where n, d_b, and d_g are integers, $d_b \leq d_g$, $S_b = \{b_1, \ldots, b_{n_b}\}$ is the set of (bad) strings, $|b_i| \geq n$, and $S_g = \{g_1, \ldots, g_{n_g}\}$ is the set of (good) strings, $|g_j| = n$, either find a string s of length n such that $D(s, b_i) \leq d_b$ for all $b_i \in S_b$, and $D(s, g_j) \geq d_g$ for all $g_j \in S_g$, or report no such a string exists.

The DSSP problem is NP-hard [80]. Recently, Deng et al. [78] (see also [79]) developed an approximation algorithm A_d for DSSP in the following sense: for a given instance $x = (n, S_b, S_g, d_b, d_g)$ for DSSP and a real number $\epsilon > 0$, *in case x is a yes-instance*, the algorithm A_d constructs a string s of length n such that $D(s, b_i) \leq d_b(1 + \epsilon)$ for all $b_i \in S_b$, and $D(s, g_j) \geq d_g(1 - \epsilon)$ for all $g_j \in S_g$. The running time of the algorithm A_d is $O(m(n_b + n_g)^{O(1/\epsilon^6)})$, where m is the size of the instance. Obviously, such an algorithm is not practical even for moderate values of the error bound ϵ.

The authors of [78] called their algorithm a “PTAS” for the DSSP problem. Strictly speaking, neither the problem DSSP nor the algorithm in [78] conforms to the standard definitions of an optimization problem and a PTAS. The DSSP problem as defined above is a decision problem with no objective function specified, and it is also not clear what precise ratio the error bound ϵ measures. We will call an algorithm in the style of the one in [78] a “PTAS-[78]” for DSSP.

Since our lower bound techniques for PTAS given in Theorem V.26 are based on the standard framework that has been widely used in the literature, we first propose an optimization version of the DSSP problem, the DSSP-OPT problem, using the standard definition of NP optimization problems. We then prove that a PTAS in the standard definition for DSSP-OPT is equivalent to a PTAS-[78] for DSSP as given in [78]. Using the systematical methods described above, we then prove that the parameterized version of DSSP-OPT is $W_l[1]$-hard, which, by Theorem V.26, gives a computational lower bound on PTAS for DSSP-OPT. As a byproduct, this also shows that it is unlikely to have a practically efficient PTAS-[78] algorithm for the DSSP problem.

Definition V.27 *The DSSP-OPT problem is a tuple (I_D, S_D, f_D, opt_D), where*

- *I_D is the set of all (yes- and no-) instances in the decision version of DSSP;*
- *For an instance $x = (n, S_b, S_g, d_b, d_g)$ in I_D, $S_D(x)$ is the set of all strings of length n;*
- *For an instance $x = (n, S_b, S_g, d_b, d_g)$ in I_D and a string $s \in S_D(x)$, the objective function value $f_D(x, s)$ is defined to be the largest non-negative integer d such that*

  (i) $d \leq d_g$;

  (ii) $D(s, b_i) \leq d_b(2 - d/d_g)$ for all $b_i \in S_b$; and

(iii) $D(s, g_j) \geq d$ *for all* $g_j \in S_g$. *If such an integer* d *does not exist, then define* $f_D(x, s) = 0$*;*

- $opt_D = \max$.

Note that for $x \in I_D$ *and* $s \in S_D(x)$*, the value* $f_D(x, s)$ *can be computed in polynomial time by checking each number* $d = 0, 1, \ldots, d_g \leq n$.

We first show that a PTAS for DSSP-OPT is equivalent to a PTAS-[78] for DSSP. Since the PTAS-[78] for DSSP is only for yes-instances of DSSP, we will concentrate on the performance of the algorithms for yes-instances of the problem DSSP.

Lemma V.28 *The DSSP-OPT problem has a PTAS of running time* $\phi(m, 1/\epsilon)$ *if and only if there is an algorithm* A_d *of running time* $\phi(m, O(1/\epsilon))$ *for DSSP that for any yes-instance of DSSP* (n, S_b, S_g, d_b, d_g) *and* $\epsilon > 0$*, constructs a string* s *of length* n *such that* $D(s, b_i) \leq d_b(1+\epsilon)$ *for all* $b_i \in S_b$*, and* $D(s, g_j) \geq d_g(1-\epsilon)$ *for all* $g_j \in S_g$.

Proof. Since $x = (n, S_b, S_g, d_b, d_g)$ is assumed to be a yes-instance of the decision problem DSSP, when x is regarded as an instance for the optimization problem DSSP-OPT, we have $opt_D(x) = d_g$.

Suppose the DSSP-OPT problem has a PTAS A_p of running time $\phi(m, 1/\epsilon)$. We show for a yes-instance $x = (n, S_b, S_g, d_b, d_g)$ and $\epsilon > 0$ how to construct a string s such that $D(s, b_i) \leq d_b(1 + \epsilon)$ for all $b_i \in S_b$, and $D(s, g_j) \geq d_g(1 - \epsilon)$ for all $g_j \in S_g$. Let $\epsilon' = \epsilon/(1 - \epsilon)$ (note that $1/\epsilon' = O(1/\epsilon)$). Apply the PTAS A_p on x and ϵ', we get a string s_p of length n such that $f_D(x, s_p) = d_p$, $opt_D(x)/d_p = d_g/d_p \leq 1 + \epsilon'$, and

$$D(s_p, b_i) \leq d_b(2 - d_p/d_g) \text{ for all } b_i \in S_b \qquad \text{and} \qquad D(s_p, g_j) \geq d_p \text{ for all } g_j \in S_g$$

Now from $d_p \geq d_g/(1 + \epsilon') = d_g(1 - \epsilon)$, we get $D(s_p, g_j) \geq d_g(1 - \epsilon)$ for all $g_j \in S_g$.

From

$$2 - d_p/d_g \leq 2 - 1/(1+\epsilon') = 1+\epsilon$$

we get $D(s_p, b_i) \leq d_b(1+\epsilon)$ for all $b_i \in S_b$. The running time of the algorithm A_p is $\phi(m, 1/\epsilon') = \phi(m, O(1/\epsilon))$. This shows that a PTAS-[78] of running time $\phi(m, O(1/\epsilon))$ for DSSP can be constructed based on the PTAS A_p for the DSSP-OPT problem.

Conversely, suppose that we have a PTAS-[78] A_d of running time $\phi(m, 1/\epsilon)$ for DSSP. We show how to construct a PTAS for the DSSP-OPT problem. For an instance $x = (n, S_b, S_g, d_b, d_g)$ of DSSP-OPT and $\epsilon > 0$, we call the algorithm A_d on x and $\epsilon' = \epsilon/(2+2\epsilon)$. By our assumption, if x is a yes-instance, then the algorithm A_d returns a string s_d of length n such that $D(s_d, b_i) \leq d_b(1+\epsilon')$ for all $b_i \in S_b$, and $D(s_d, g_j) \geq d_g(1-\epsilon')$ for all $g_j \in S_g$. We first consider the value $f_D(x, s_d)$ for DSSP-OPT. Let $d = d_g - \lceil \epsilon' d_g \rceil$. Then for each good string g_j, we have

$$D(s_d, g_j) \geq d_g(1-\epsilon') = d_g - \epsilon' d_g \geq d_g - \lceil \epsilon' d_g \rceil = d$$

and since $d = d_g - \lceil \epsilon' d_g \rceil \leq d_g - \epsilon' d_g = d_g(1-\epsilon')$, for each bad string b_i,

$$D(s_d, b_i) \leq d_b(1+\epsilon') = d_b(2-(1-\epsilon')) \leq d_b(2 - d/d_g)$$

By the definition of the function $f_D(x, s_d)$, we have $f_D(x, s_d) \geq d = d_g - \lceil \epsilon' d_g \rceil$.

Now consider the ratio $opt_D(x)/f_D(x, s_d)$ for the string s_d. If $\epsilon' d_g < 0.5$, then (note that $d_b \leq d_g$)

$$D(s_d, b_i) \leq d_b(1+\epsilon') < d_b + 0.5 \quad \text{and} \quad D(s_d, g_j) \geq d_g(1-\epsilon') > d_g - 0.5$$

Since all $D(s_d, b_i)$, d_b, $D(s_d, g_j)$, and d_g are integers, we have $D(s_d, b_i) \leq d_b = d_b(2 - d_g/d_g)$ for all $b_i \in S_b$, and $D(s_d, g_j) \geq d_g$ for all $g_j \in S_g$. Therefore, we have $f_D(x, s_d) = d_g$ and $opt(x)/f_D(x, s_d) = 1$. On the other hand, if $\epsilon' d_g \geq 0.5$, then

$d_g - \lceil \epsilon' d_g \rceil \geq d_g - 2\epsilon' d_g$, and we have

$$opt(x)/f_D(x, s_d) \leq d_g/(d_g - \lceil \epsilon' d_g \rceil) \leq d_g/(d_g - 2\epsilon' d_g) = 1/(1 - 2\epsilon') = 1 + \epsilon$$

Therefore, in all cases, the string s_d produced by the algorithm A_d is a solution of approximation ratio $1 + \epsilon$ for the instance x of DSSP-OPT. Again, the running time of the algorithm is dominated by that of A_d, which is bounded by $\phi(m, 1/\epsilon') = \phi(m, O(1/\epsilon))$.

This completes the proof of the lemma. □

Lemma V.28 shows that a PTAS-[78] for the problem DSSP is also a PTAS in the standard definition for the optimization problem DSSP-OPT.

Now using the standard parameterization of optimization problems, we can study the parameterized complexity of the problem DSSP-OPT$_{\geq}$.

Lemma V.29 *The parameterized problem DSSP-OPT$_{\geq}$ is $W_l[1]$-hard.*

Proof. We prove the lemma by an fpt$_l$-reduction from the $W_l[1]$-hard problem Dominating Set to the DSSP-OPT$_{\geq}$ problem (see Theorem V.24).

Let (G, k) be an instance of the Dominating Set problem. Suppose that the graph G has n vertices v_1, …, v_n. Denote by $vec(v_i)$ the binary string of length n in which all bits are 0 except the i-th bit is 1. The instance $x_G = (n', S_b, S_g, d_b, d_g)$ for DSSP-OPT is constructed as follows: $n' = n + 5$, S_g consists of a single string $g_0 = 0^{n+5}$, $d_b = k - 1$, and $d_g = k + 3$.

The bad string set $S_b = \{b_1, \ldots, b_n\}$ consists of n strings, where b_i corresponds to the vertex v_i in G. Suppose the neighbors of the vertex v_i in G are v_{i_1}, …, v_{i_r}, then the string b_i takes the form

$$vec(v_i) \cdot 02220 \cdot vec(v_i) \cdot 00000 \cdot vec(v_{i_1}) \cdot 02220 \cdot vec(v_{i_1}) \cdot$$
$$\cdot 00000 \cdot \cdots \cdot 00000 \cdot vec(v_{i_r}) \cdot 02220 \cdot vec(v_{i_r})$$

where the dots "$\cdot$" stand for string concatenations. It is easy to see that the size of x_G is bounded by a polynomial of the size of the graph G. Finally, we set the parameter $k' = k + 3$. Thus, (x_G, k') makes an instance for the DSSP-OPT$_\geq$ problem.

We prove that (G, k) is a yes-instance for Dominating Set if and only if (x_G, k') is a yes-instance for DSSP-OPT$_\geq$. Suppose the graph G has a dominating set H of k vertices. Let $vec(H)$ be the binary string of length n whose h-th bit is 1 if and only if $v_h \in H$. Now consider the string $s = vec(H) \cdot 02220$. Clearly $D(s, g_0) = k + 3 = d_g$. For each bad string b_i, since H is a dominating set, either $v_i \in H$ or a vertex $v_j \in H$ is a neighbor of v_i. If $v_i \in H$ then the substring $b_i' = vec(v_i) \cdot 02220$ in b_i satisfies $D(s, b_i') = k - 1$, and if a vertex $v_j \in H$ is a neighbor of v_i, then the substring $b_i' = vec(v_j) \cdot 02220$ in b_i satisfies $D(s, b_i') = k - 1$. This verifies that $D(s, b_i) = k - 1 = d_b(2 - d_g/d_g)$ for all $1 \leq i \leq n$. Thus, for the string s, we have $f_D(x_G, s) = opt_D(x_G) = d_g = k + 3 \geq k'$. In consequence, (x_G, k') is a yes-instance of DSSP-OPT$_\geq$.

Conversely, suppose (x_G, k') is a yes-instance for the DSSP-OPT$_\geq$ problem. Then there is a string s of length $n + 5$ such that $f_D(x_G, s) = d \geq k' = k + 3$. By the definition, $f_D(x_G, s) \leq d_g = k+3$. Thus, we must have $d = k+3$. From the definition of the integer d, we have $D(s, g_0) \geq d = k+3$, and $D(s, b_i) \leq d_b(2-d/d_g) = d_b = k-1$ for all bad strings b_i. Since $g_0 = 0^{n+5}$ and $D(s, g_0) \geq k+3$, s has at least $k+3$ "non-0" bits. On the other hand, it is easy to see that each substring of length $n+5$ in any bad string b_i contains at most 4 "non-0" bits. Since $D(s, b_i) \leq k - 1$ for each bad string b_i, the string s should not contain more than $k + 3$ "non-0" bits. Thus, the string s

has exactly $k+3$ "non-0" bits. Now consider any substring b_i' of length $n+5$ in a bad string b_i such that $D(s, b_i') \leq k-1$. The substring b_i' must contain "222": otherwise b_i' has at most three "non-0" bits so $D(s, b_i') \leq k-1$ would not be possible. If the substring"222" in b_i' does not match three "2"'s in s, then s has at least k "non-0" bits in other places while b_i' has only one "non-0" bit in other place, so $D(s, b_i') \leq k-1$ would not be possible. Thus, the string s must contain the substring "222", which matches the substring "222" in b_i'. Finally, observe that we can always assume that the string s ends with "02220" – otherwise we simply cyclically shift the string s to move the substring "02220" to the end. Note if $D(s, b_i') \leq k-1$ and b_i' is a substring in a segment "$00000 \cdot vec(v_j) \cdot 02220 \cdot vec(v_j) \cdot 00000$" in the bad string b_i, then after shifting s, we must have $D(s, b_i'') \leq k-1$, where $b_i'' = vec(v_j) \cdot 02220$. Therefore, if s is a solution to the instance (x_G, k'), then so is the string after the cyclic shifting.

Thus, the string s can be assumed to have the form $s' \cdot 02220$, where s' is a string of length n, with exactly k "non-0" bits. Suppose that the j_1-th, j_2-th, $\ldots$, and j_k-th bits of s' are "non-0". We claim that the vertex set $H_s = \{v_{j_1}, \ldots, v_{j_k}\}$ makes a dominating set of k vertices for the graph G. In fact, for any bad string b_i, let b_i' be a substring of length $n+5$ in b_i such that $D(s, b_i') \leq k-1$. According to the above discussion, b_i' must be of the form $vec(v_j) \cdot 02220$, where either $v_j = v_i$ or v_j is a neighbor of v_i. The only "non-0" bit in $vec(v_j)$ is the j-th bit, and j must be among $\{j_1, \ldots, j_k\}$ – otherwise $D(vec(v_j), s')$ is at least $k+1$. Therefore, if $v_i = v_j$ then $v_i \in H_s$, and if v_j is a neighbor of v_i, then v_i is adjacent to the vertex v_j in H_s. This proves that H_s is a dominating set of k vertices in G, and that (G, k) is a yes-instance for Dominating Set.

This completes the proof that the problem Dominating Set is fpt$_l$-reducible to the problem DSSP-OPT$_\geq$. In consequence, DSSP-OPT$_\geq$ is $W_l[1]$-hard. □

We remark that the problem Dominating Set is $W[2]$-hard under the regular fpt-reduction [11]. Therefore, the proof of Lemma V.29 actually shows that the DSSP-OPT$_{\geq}$ problem is $W[2]$-hard. This improves the result in [80], which proved that the problem is $W[1]$-hard.

From Lemma V.29 and Theorem V.26, we get immediately

Theorem V.30 *Unless all SNP problems are solvable in subexponential time, the optimization problem DSSP-OPT has no PTAS of running time $f(1/\epsilon)m^{o(1/\epsilon)}$ for any function f.*

By Lemma V.28, a PTAS-[78] of running time $f(1/\epsilon)m^{o(1/\epsilon)}$ for DSSP would imply a PTAS of running time $f'(1/\epsilon)m^{o(1/\epsilon)}$ for DSSP-OPT for a function f'. Therefore, Theorem V.30 also implies that any PTAS-[78] for DSSP cannot run in time $f(1/\epsilon)m^{o(1/\epsilon)}$ for any function f. Thus essentially, no PTAS-[78] for DSSP can be practically efficient even for moderate values of the error bound ϵ. To the authors' knowledge, this is the first time a specific lower bound is derived on the running time of a PTAS for an NP-hard problem.

Theorem V.30 also demonstrates the usefulness of our techniques. In most cases, computational lower bounds and inapproximability of optimization problems are derived based on approximation ratio-preserving reductions [2], by which if a problem Q_1 is reduced to another problem Q_2, then Q_2 is at least as hard as Q_1. In particular, if Q_1 is reduced to Q_2 under an approximation ratio-preserving reduction, then the approximability of Q_2 is at least as difficult as that of Q_1. Therefore, the intractability of an "easier" problem in general cannot be derived using such a reduction from a "harder" problem. On the other hand, our computational lower bound on DSSP-OPT was obtained by a linear fpt-reduction from Dominating Set. It is well-known that Dominating Set has no polynomial time approximation algorithms of constant ratio [2], while DSSP-OPT has PTAS. Thus, from the viewpoint of approximability, Dominating Set is much harder than DSSP-OPT, and our linear fpt-reduction reduces a harder problem to an easier problem. This hints that our approach for deriving computational lower bounds *cannot* be simply replaced by the standard approaches based on approximation ratio-preserving reductions.

G. Comments

In this chapter, based on parameterized complexity theory, we developed new techniques for deriving computational lower bounds for well-known NP-hard problems. We started by establishing the computational lower bounds for the generic parameterized problems WCS$^*[t]$ for $t \geq 2$ and WCNF 2-SAT$^-$. We showed that for any integer $t \geq 2$, an $f(k)n^{o(k)}m^{O(1)}$-time algorithm for WCS$^*[t]$ for any function f would collapse the $(t-1)$-st level $W[t-1]$ to the bottom level FPT in the fixed-parameter intractability hierarchy, the W-hierarchy, and that an $f(k)m^{o(k)}$-time algorithm for WCNF 2-SAT$^-$ would imply subexponential time algorithms for all problems in SNP. Based on these generic results, we introduced the concept of linear fpt-reductions, and used it to derive tight computational lower bounds for many well-known NP-hard problems. Obviously, the list of the problems we have given here is far from being exhaustive. This new technique should serve as a very powerful tool for deriving strong computational lower bounds for other intractable problems. Moreover, we demonstrated how our techniques can be used to derive strong computational lower bounds on polynomial time approximation schemes for NP-hard problems. This seems to open a new direction for the study of computational lower bounds on the approximability of NP-hard optimization problems.

CHAPTER VI

SUMMARY AND FUTURE RESEARCH

In this dissertation, we took a structural approach in designing efficient parameterized algorithms for a set of well-known NP-hard problems and proving strong lower bounds for some others. Many of the techniques introduced in this dissertation have the potential to deliver further improved algorithms and tighter lower bounds in the future.

A. Thesis Summary

We designed new algorithms for some of the most well studied NP-hard problems. In particular, we presented a new algorithm for Vertex Cover that is simpler than most of the previous algorithms, yet has better performance than all of them, including those using exponential space. For Vertex Cover on graphs with degree bounded by three, we presented a still better algorithm by introducing a more global way of analyzing the search tree. For other graph problems on graphs with constrained genus, we showed that the graph genus is the major factor in determining their parameterized complexity, approximability, and subexponential computability. Of particular interest in this exposition are the new techniques introduced in designing the above algorithms.

We developed a set of new techniques that allowed us to provide convincing evidence that it is unlikely to do much better than the brute-force algorithm in solving some of the well-known intractable problems, including Clique, Dominating Set, Hitting Set, Set Cover, and Independent Set. The same techniques were also applied to derive lower bounds on the running time of approximation algorithms for many practical problems.

B. Future Work

1. Improve the Upper Bounds for Independent Set

Vertex Cover and Independent Set are closely related. In fact, if C is a minimum vertex cover of a graph G, then $V - C$ constitutes a maximum independent set. The rich structural properties and techniques that we have developed for Vertex Cover are naturally applicable for finding maximum independent set. Exactly algorithms for Maximum Independent Set have been extensively studied over the years [46, 28, 77, 45]. However, it seems to be difficult to make further progress on this problem. In fact, the number of cases in [77] has reached a point where much of the most detailed analysis has to be done by computers.

Our parameterized algorithm for Vertex Cover is in its current form an algorithm for Independent Set. By a simple relation between the number of vertices in a graph and the size of its vertex cover, the algorithm induces improvement on the upper bound for the Independent Set problem on graphs of degree bounded by 6. Note that most of the techniques for Vertex Cover, including *tuple*, *struction*, and *general folding* work quite well for Independent Set. A more careful analysis or some modified form of the current algorithm may yield an improved upper bound for Independent Set on general graphs.

2. Tighter Analysis of Search Tree

Until recently, most of the efficient parameterized algorithms are based on refined branch-and-bound method. This method is generally efficient in practice. However, in order to prove bounds on the running time, a lot of special cases have to made just for the propose of the proof. Furthermore, in analyzing a search tree representing a branch-and-bound algorithm, the common approach is to prove that *every* branching satisfies certain condition, which is difficult and the result is usually far from being tight.

Our method of "almost-global" amortized analysis on paths of the search tree is the first step toward a "global" analysis of the search tree. This new method opens a new direction in the analysis of the running time of exact algorithms for NP-hard problems that are based on branch-and-bound. Instead of looking at sophisticated algorithms and deriving an easy but conservative upper bound on the size of the search tree, we can consider instead very simple and intuitive algorithms, and perform an amortized analysis that reflects more closely the actual size of the search tree. We believe that this method of analysis is applicable to a variety of NP-hard problems. A natural extension of this method is to apply global analyze to the search tree as a whole, thus balance all branchings in the entire search tree. This could potentially yield much better bound without making the algorithm complex and the analysis tedious.

3. Techniques for Designing Efficient Parameterized Algorithms

Parameterized computation was introduced under practical motivation [11]. That is to design efficient algorithms for problems arising in real applications. While it is important to proving better upper bounds for parameterized problems in theory, emphasis should be placed on performances of the parameterized algorithms in practice. For example, a common technique for designing parameterized algorithms is a randomized method, called the *color-coding* method proposed by Alon, Yuster and Zwick [21]. Some recent efforts are focused on making this technique more practical in real applications [85].

Another interesting technique for designing efficient parameterized algorithms is the "kernelization" technique, which has been used in designing parameterized algorithms for Vertex Cover in Proposition II.1. Smaller kernels for parameterized problems have been extensively studied in recent years. For example, Alber et al. [71] presented a data reduction algorithm, and showed that it reduces the Planar Dominating Set problem to a kernel of size bounded by $335k$. This kernel was further reduced to 67 [86]. We believe that smaller kernels of NP-hard problems not only induce fixed-parameter tractable algorithms for them, but could also improve other types of algorithms for them by significantly reducing the size of the instances.

4. Graph Genus and Computational Complexity

Our results showed that a class of NP-hard graph problems, including some very well-known ones, become more tractable on lower genus graphs. It is interesting to compare our results to the results in [87], which shows that certain other NP-hard problems become more tractable on dense graphs, for which the graph genus is necessarily high. We notice that the problems studied in [87] are most graph cutting problems, such as Max-Cut, and Graph-Bisection, while problems studied in this dissertation are vertex subset problems. A systematical study of the difference between these two kinds of NP-hard problems looks rather appealing.

Our results on the genus threshold for fixed parameter tractability and subexponential time computability (Sections B and C of Chapter IV) are tight. Our results on polynomial time approximation schemes (Section D of Chapter IV), however, have a gap between $o(n/\log n)$ and $\Omega(n)$ on the genus bound. According to [68], when the graph genus is $o(n)$, there is a set of $o(n)$ vertices whose removal results in a planar graph. However, no algorithm is known that efficiently constructs such a set. It should be interesting and seems to be possible to close the above genus gap.

5. Stronger Lower Bounds

We have provided convincing evidence that it is unlikely to do much better than the brute-force algorithm in solving some of the well-known intractable problems. However, there is still a gap between the known upper bounds and the proven lower bounds. It will be interesting if we could prove any lower bounds beyond the subexponential running time, based on parameterized complexity hypothesis. For example, whether or not we could prove that Clique cannot be solved in time n^{ck} for some constant $c \geq 0$ unless an unlikely collapse occurs in parameterized complexity theory.

We have also proved lower bounds on the running time of approximation algorithms for some practical problems. An interesting open problem is to study whether or not parameterized complexity hypothesis will lead to inapproximability results. Currently, we are only able to prove that some problems don't have certain approximation schemes under plausible assumptions.

We are also interested in establishing lower bounds and inapproximability results on weaker assumptions. For example, currently, we are only able to prove that Vertex Cover is not solvable in subexponential time unless all problems in the class MAXSNP is solvable in subexponential time. An obvious improvement is to establish the same lower bonds on the weaker assumption that W[1] $\neq$ FPT.

Recently, some efforts have been focused on finding subexponential approximation algorithms of improved ratio for problems such as Vertex Cover or proving that such algorithms do not exist. Lower bounds or inapproximability results on such problems would shed light on the inherent difficulties in approximating these problems.

REFERENCES

[1] Z. Michalewicz and D. B. Fogel, eds., *How to Solve It: Modern Heuristics.* Berlin, Germany: Springer, 2000.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Berlin, Germany: Springer-Verlag, 1999.

[3] R. Motwani and P. Raghavan, eds., *Randomized Algorithms.* New York: Cambridge University Press, 1995.

[4] C. Roth-Korostenskyg, *Algorithms for Building Multiple Sequence Alignments and Evolutionary Trees.* PhD thesis, Swiss Federal Institute of Technology in Zurich, 2000.

[5] U. Stege, *Resolving Conflicts from Problems in Computational Biology.* PhD thesis, Swiss Federal Institute of Technology in Zurich, 2000.

[6] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness.* San Francisco, CA: Freeman, 1979.

[7] I. Dinur and S. Safra, "The importance of being biased," in *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC'02)*, Montréal, Canada, pp. 33–42, May 2002.

[8] DIMACS, *DIMACS Workshop on Faster Exact Algorithms for NP-Hard Problem.* February 2000. available at http://dimacs.rutgers.edu/Workshops/Faster/.

[9] D. S. Johnson and M. A. Tricks, eds., *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenges.* DIMACS Series on Discrete Mathematics and Theoretical Computer Science 26, Providence, RI: American Mathematical Society, 1996.

[10] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Computing*, vol. 44, pp. 279–303, 1990.

[11] R. G. Downey and M. Fellows, *Parameterized Complexity.* New York: Springer, 1999.

[12] J. Chen, I. A. Kanj, and W. Jia, "Vertex cover: Further observations and further improvements," *Journal of Algorithms*, vol. 41, pp. 280–301, 2001.

[13] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. Taillon, "Solving large FPT problems on coarse grained parallel machines," *Journal of Computer and System Sciences*, vol. 67, pp. 691–701, 2003.

[14] L. Cai and D. Juedes, "On the existence of subexponential-time parameterized algorithms," *Journal of Computer and System Sciences*, vol. 67, no. 4, pp. 789–807, 2003.

[15] R. Impagliazzo, R. Paturi, and F. Zane, "Which problems have strongly exponential complexity?," *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 512–530, 2001.

[16] D. S. Johnson and M. Szegedy, "What are the least tractable instances of max. independent set?," in *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, Baltimore, MD, pp. 927–928, January 1999.

[17] C. H. Papadimitriou and M. Yannakakis, “On the complexity of database queries,” *Journal of Computer and System Sciences*, vol. 58, pp. 407–427, 1999.

[18] M. Cesati and L. Trevisan, “On the efficiency of polynomial time approximation schemes,” *Information Processing Letters*, vol. 64, pp. 165–171, 1997.

[19] U. Feige and J. Kilian, “Nonconstructive tools for proving polynomial time decidability,” *Journal of ACM*, vol. 35, no. 3, pp. 727–739, 1988.

[20] H. L. Bodlaender, “A linear time algorithm for finding tree-decompositions of small treewidth,” *SIAM Journal on Computing*, vol. 25, pp. 1305–1317, 1996.

[21] N. Alon, R. Yuster, and U. Zwick, “Color-coding,” *Journal of the ACM*, vol. 42, pp. 844–856, 1995.

[22] R. Balasubramanian, M. R. Fellows, and V. Raman, “An improved fixed parameter algorithm for vertex cover,” *Information Processing Letters*, vol. 65, pp. 163–168, 1998.

[23] J. Chen, L. Liu, and W. Jia, “Improvement on vertex cover for low-degree graphs,” *Networks*, vol. 35, pp. 253–259, 2000.

[24] R. Downey and M. Fellows, “Fixed-parameter tractability and completeness,” *Congressus Numerantium*, vol. 87, pp. 161–187, 1992.

[25] R. Niedermeier and P. Rossmanith, “Upper bounds for vertex cover further improved,” in *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS’99)*, LNCS 1563, Trier, Germany, pp. 561–570, March 1999.

[26] R. Niedermeier and P. Rossmanith, “On efficient fixed-parameter algorithms for weighted vertex cover,” *Journal of Algorithms*, vol. 47, pp. 63–77, 2003.

[27] U. Stege and M. Fellows, “An improved fixed-parameter-tractable algorithm for vertex cover,” Research Report 318, Swiss Federal Institute of Technology in Zurich, 1999.

[28] J. M. Robson, “Algorithms for maximum independent sets,” *Journal of Algorithms*, vol. 7, no. 3, pp. 425–440, 1986.

[29] C. Ebengger, P. Hammer, and D. de Werra, “Pseudo-Boolean functions and stability of graphs,” *Annals of Discrete Mathematics*, vol. 19, pp. 83–98, 1984.

[30] L. S. Chandran and F. Grandonn, “Refined memorisation for vertex covers,” *Information Processing Letters*, vol. 93, pp. 125–131, 2004.

[31] J. Chen, I. Kanj, L. Perkovic, E. Sedgwick, and G. Xia, “Genus characterizes the complexity of graph problems: Some tight results,” in *Proc. 30th International Colloquium on Automata, Languages, and Programming (ICALP’03)*, LNCS 2719, Eindhoven, The Netherlands, pp. 845–856, June-July 2003.

[32] J. Alber, H. Fernau, and R. Niedermeier, “Parameterized complexity: Exponential speedup for planar graph problems,” *Journal of Algorithms*, vol. 52, pp. 26–56, 2004.

[33] J. Ellis, H. Fan, and M. Fellows, “The dominating set problem is fixed parameter tractable for graphs of bounded genus,” *Journal of Algorithms*, vol. 52, pp. 152–168, 2004.

[34] R. Diestel, *Graph Theory.* Graduate Texts in Mathematics 173, 2nd ed., New York: Springer-Verlag, 2000.

[35] G. Woeginger, “Exact algorithms for NP-hard problems: A survey,” in *Proc. 5th International Workshop on Combinatorial Optimization - Eureka! You shrink!*,

LNCS, 2570, Aussois, France, pp. 185–207, March 2001.

[36] J. F. Buss and J. Goldsmith, “Nondeterminism within P,” *SIAM Journal on Computing*, vol. 22, pp. 560–572, 1993.

[37] L. Cai and D. Juedes, “On the existence of subexponential parameterized algorithms,” *Journal of Computer and System Sciences*, vol. 67, no. 4, pp. 789–807, 2003.

[38] R. Impagliazzo, R. Paturi, and F. Zane, “Which problems have strongly exponential complexity?,” *Journal of Computer and System Sciences*, vol. 63, no. 4.

[39] G. Nemhauser and L. Trotter, “Vertex packing: Structural properties and algorithms,” *Mathematical Programming*, vol. 8, pp. 232–248, 1975.

[40] M. Chlebik and J. Chlebikova, “Crown reductions for the minimum weighted vertex cover problem,” in *Electronic Colloquium on Computational Complexity (ECCC), Report No. 101*, 2004. available at http://eccc.uni-trier.de/eccc-reports/2004/TR04-101/index.html.

[41] M. Fellows, “Blow-ups, win/win’s, and crown rules: Some new directions in FPT,” in *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG’03*, LNCS 2880, Elspeet, the Netherlands, pp. 1–12, June 2003.

[42] J. Chen, I. Kanj, and G. Xia, “Labeled search trees and amortized analysis: Improved upper bounds for NP-hard problems,” in *Proc. 14th International Symposium on Algorithms and Computation (ISAAC’03)*, LNCS 2906, Kyoto, Japan, pp. 148–157, December 2003.

[43] R. Niedermeier and P. Rossmanith, “A general method to speed up fixed-parameter-tractable algorithms,” *Information Processing Letters*, vol. 73:3-4,

pp. 125–129, 2000.

[44] R. E. Tarjan and A. E. Trojanowski, “Finding a maximum independent set,” *SIAM Journal on Computing*, vol. 6, pp. 537–546, 1977.

[45] R. Beigel, “Finding maximum independent sets in sparse and general graphs,” in *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithm (SODA'99)*, Baltimore, MD, pp. 856–857, January 1999.

[46] T. Jian, “An $o(2^{0.304n})$ algorithm for solving the maximum independent set problem,” *IEEE Transactions on Computers*, vol. 35, no. 4, pp. 847–851, 1986.

[47] R. Niedermeier and P. Rossmanith, “Upper bounds for vertex cover further improved,” in *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, LNCS 1563, Trier, Germany, pp. 561–570, March 1999.

[48] M. Shindo and E. Tomita, “A simple algorithm for finding a maximum clique and its worst-case time complexity,” *Sys. and Comp. in Japan*, vol. 21, pp. 1–13, 1990.

[49] W. T. Tutte, ed., *Graph Theory.* Menlo Park, CA: Addison-Wesley Publishing Company, 1984.

[50] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley Publishing Company, 1974.

[51] J. E. Hopcroft and R. E. Tarjan, “Dividing a graph into triconnected components,” *SIAM Journal on Computing*, vol. 2, pp. 135–158, 1973.

[52] J. Alber, H. Bodlaender, H. Ferneau, and R. Niedermeier, “Fixed parameter algorithms for dominating set and related problems on planar graphs,” *Algorithmica*, vol. 33, pp. 461–493, 2002.

[53] B. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *Journal of the ACM*, vol. 41, pp. 153–180, 1994.

[54] R. Lipton and R. Tarjan, "Applications of a planar separator theorem," *SIAM Journal on Computing*, vol. 9, pp. 615–627, 1980.

[55] D. Demaine, F. Fomin, M. Hajiaghayi, and D. Thilikos, "Subexponential parameterized algorithms on graphs of bounded-genus and h-minor-free graphs," in *Proc. 15th ACM-SIAM Symp. Discrete Algorithm (SODA'04)*, New Orleans, LA, pp. 830–839, January 2004.

[56] F. Fomin and D. Thilikos, "Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up," in *Proc. 31th International Colloquium on Automata, Languages, and Programming (ICALP'04)*, LNCS 3142, Turku, Finland, pp. 581–592, July 2004.

[57] H. Fleischner, S. Földes, and S. Szeider, "Remarks on the concept of robust algorithm," Research Report 26, RUTCOR Research Report, RUTCOR, Piscataway, NJ, 2001.

[58] J. Gross and T. Tucker, *Topological Graph Theory.* New York: Wiley-Interscience, 1987.

[59] J. Chen, "Algorithmic graph embeddings," *Theoretical Computer Science*, vol. 181, pp. 247–266, 1987.

[60] J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, "Tight lower bounds for certain parameterized NP-hard problems," in *Proc. 19th Annual IEEE Conference on Computational Complexity (CCC'04)*, Amherst, MA, pp. 150–160, June 2004.

[61] J. Chen, X. Huang, I. Kanj, and G. Xia, “Linear FPT reductions and computational lower bounds,” in *Proc. 34th ACM Symposium on theory of Computing (STOC’04)*, Chicago, pp. 212–221, June 2004.

[62] C. Thomassen, “The graph genus problem is NP-complete,” *Journal of Algorithms*, vol. 10, pp. 568–576, 1989.

[63] J. Chen, S. Kanchi, and A. Kanevsky, “A note on approximating graph genus,” *Information Processing Letters*, vol. 61, pp. 317–322, 1997.

[64] D. Demaine, M. Hajiaghayi, and D. Thilikos, “Exponential speedup of fixed-parameter algorithms on $k_{3,3}$-free or k_5-free graphs,” in *Proc. 13th Int. Symp. Algorithms & Computation (ISAAC’02)*, LNCS 2518, Vancouver, Canada, pp. 262–273, November 2004.

[65] F. Fomin and D. Thilikos, “Dominating sets in planar graphs: Branch-width and exponential speed-up,” in *Proc. 14th ACM-SIAM Symp. Discrete Algorithms (SODA’03)*, Baltimore, MD, pp. 168–177, January 2003.

[66] I. Kanj and L. Perkovic, “Improved parameterized algorithms for planar dominating set,” in *27th International Symposium on Mathematical Foundations of Computer Science (MFCS’02)*, Warszawa, Poland, pp. 399–410, August 2002.

[67] C. H. Papadimitriou and M. Yannakakis, “Optimization, approximation and complexity classes,” *Journal of Computer and System Sciences*, vol. 43, pp. 425–440, 1991.

[68] H. Djidjev and S. Venkatesan, “Planarization of graphs embedded on surfaces,” in *Proc. 21st International Workshop on Graph-Theoretic Concepts in Computer Science (WG’95)*, LNCS 1017, Aachen, Germany, pp. 62–72, June 1995.

[69] J. Chen, I. Kanj, and G. Xia, “A note on subexponential parameterized complexity,” Research Report 23, School of Computer Science, Telecommunications and Information Systems, DePaul University, Chicago, 2003.

[70] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, “Proof verification and hardness of approximation problems,” *Journal of the ACM*, vol. 45, pp. 501–555, 1998.

[71] J. Alber, M. Fellows, and R. Niedermeier, “Polynomial time data reduction for dominating set,” *Journal of the ACM*, vol. 51, pp. 363–384, 2004.

[72] D. Hochbaum, “Efficient bounds for the stable set, vertex cover and set packing problems,” *Discrete Applied Mathematics*, vol. 6, pp. 243–254, 1983.

[73] K. A. Abrahamson, R. G. Downey, and M. R. Fellows, “Fixed-parameter tractability and completeness iv: On completeness for W[P] and PSPACE analogs,” *Annals of Pure and Applied Logic*, vol. 73, pp. 235–276, 1995.

[74] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, “Fixed parameter algorithms for dominating set and related problems on planar graphs,” *Algorithmica*, vol. 33, pp. 461–493, 2002.

[75] D. Coppersmith and S. Winograd, “Matrix multiplication via arithmetic progression,” *Journal of Symbolic Computation*, vol. 9, pp. 251–280, 1990.

[76] J. Nešetřil and S. Poljak, “On the complexity of the subgraph problem,” *Commentationes Mathematicae Universitatis Carolinae*, vol. 26, pp. 415–419, 1985.

[77] J. M. Robson, “Finding a maximum independent set in time $o(2^{n/4})$?,” Research Report 1251, LaBRI, Universite Bordeaux I, Talence Cedex, France, 2001.

[78] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang, "A PTAS for distinguishing (sub)string selection," in *Proc. 29th International Colloquium on Automata, Languages, and Programming (ICALP'02)*, LNCS 2518, Málaga, Spain, pp. 740–751, July 2002.

[79] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang, "Genetic design of drugs without side-effects," *SIAM Journal on Computing*, vol. 32, pp. 1073–1090, 2003.

[80] J. Gramm, J. Guo, and R. Niedermeier, "On exact and approximation algorithms for distinguishing substring selection," in *Proc. 14th International Symposium on Fundamentals of Computation Theory, (FCT'03)*, Malmö, Sweden, pp. 195–209, August 2003.

[81] J. Chen, "Characterizing parallel hierarchy by reducibilities," *Information Processing Letters*, vol. 39, pp. 303–307, 1991.

[82] R. G. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez, and F. Rosamond, "Cutting up is hard to do: The parameterized complexity of k-cut and related problems," *Electronic Notes in Theoretical Computer Science*, vol. 78, pp. 205–218, 2003.

[83] U. Feige and J. Kilian, "On limited versus polynomial nondeterminism," *Chicago Journal of Theoretical Computer Science*, vol. 1, pp. 1–20, 1997.

[84] C. H. Papadimitriou and M. Yannakakis, "On limited nondeterminism and the complexity of VC dimension," *Journal of Computer and System Sciences*, vol. 53, pp. 161–170, 1996.

[85] J. Chen, "Color coding revised." private communications, April 2005.

[86] J. Chen, H. Fernau, I. Kanj, and G. Xia, "Parametric duality and kernelization: Lower bounds and upper bounds on kernel size," in *Proc. 22nd International Symposium on Theoretical Aspects of Computer Science (STACS'05)*, Stuttgart, Germany, pp. 269–280, February 2005.

[87] S. Arora, D. Karger, and M. Karpinski, "Polynomial time approximation schemes for dense instances of NP-hard problems," in *Proc. 27th Annual ACM Symp. on Theory of Computing (STOC'95)*, Las Vegas, pp. 284–293, May-June 1995.

VITA

The author, Ge Xia, received his B.Arch. degree in architecture from Tongji University, China in August 1998, and his M.S. degree in architecture from Texas A&M University in August 2001.

Mr. Xia received his Ph.D. degree in computer science from Texas A&M University in August 2005. His thesis advisor is Dr. Jianer Chen. Mr. Xia's research interests include Design and Analysis of Algorithms, Graph Theory, Computational Optimization, Bioinformatics and Complexity Theory.

Mr. Xia's current address is: Department of Computer Science, Lafayette College, Easton, PA 18042, USA. Email: gexia@cs.lafayette.edu