PARAMETERIZED COMPLEXITY AND POLYNOMIAL-TIME APPROXIMATION SCHEMES

A Dissertation

by

XIUZHEN HUANG

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

December 2004

Major Subject: Computer Science

PARAMETERIZED COMPLEXITY AND

POLYNOMIAL-TIME APPROXIMATION SCHEMES

A Dissertation

by

XIUZHEN HUANG

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

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Jianer Chen
(Chair of Committee)

Donald K. Friesen
(Member)

Sing-Hoi Sze
(Member)

Catherine Yan
(Member)

Valerie E. Taylor
(Head of Department)

December 2004

Major Subject: Computer Science

ABSTRACT

Parameterized Complexity and

Polynomial-Time Approximation Schemes. (December 2004)

Xiuzhen Huang, B.S., Shandong University;

M.S., Shandong University

Chair of Advisory Committee: Dr. Jianer Chen

According to the theory of NP-completeness, many problems that have important real-world applications are NP-hard. This excludes the possibility of solving them in polynomial time unless P=NP. A number of approaches have been proposed in dealing with NP-hard problems, among them are approximation algorithms and parameterized algorithms. The study of approximation algorithms tries to find good enough solutions instead of optimal solutions in polynomial time, while parameterized algorithms try to give exact solutions when a natural parameter is small.

In this thesis, we study the structural properties of parameterized computation and approximation algorithms for NP optimization problems. In particular, we investigate the relationship between parameterized complexity and polynomial-time approximation scheme (PTAS) for NP optimization problems.

We give nice characterizations for two important subclasses in PTAS: Fully Polynomial Time Approximation Scheme (FPTAS) and Efficient Polynomial Time Approximation Scheme (EPTAS), using the theory of parameterized complexity. Our characterization of the class FPTAS has its advantages over the former characterizations, and our characterization of EPTAS is the first systematic investigation of this new but important approximation class.

We develop new techniques to derive strong computational lower bounds for

certain parameterized problems based on the theory of parameterized complexity. For example, we prove that unless an unlikely collapse occurs in parameterized complexity theory, the CLIQUE problem could not be solved in time $O(f(k)n^{o(k)})$ for any function f. This lower bound matches the upper bound of the trivial algorithm that simply enumerates and checks all subsets of k vertices in the given graph of n vertices.

We then extend our techniques to derive computational lower bounds for PTAS and EPTAS algorithms of NP optimization problems. We prove that certain NP optimization problems with known PTAS algorithms have no PTAS algorithms of running time $O(f(1/\epsilon)n^{o(1/\epsilon)})$ for any function f. Therefore, for these NP optimization problems, although theoretically they can be approximated in polynomial time to an arbitrarily small error bound ϵ, they have no practically effective approximation algorithms for small error bound ϵ. To our knowledge, this is the first time such lower bound results have been derived for PTAS algorithms. This seems to open a new direction for the study of computational lower bounds on the approximability of NP optimization problems.

To my husband and our daughter (four year old)

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Jianer Chen, for his tremendous help, guidance and encouragement during my study in Texas A&M University. Dr. Chen has led me into this exciting and promising research area and guided me throughout my PhD research. His excellence in both teaching and research in computer science makes him a great example for me to follow in my academic career.

A special thanks to Dr. Donald K. Friesen for his kindness to the students and excellent work in our department.

A special thanks to Dr. Sing-Hoi Sze for his excellent course on "Special Topics in Computational Biology".

A special thanks to Dr. Catherine Yan, who gave me advice on my thesis with her one-month-old son sleeping in her arms.

I would like to thank all my committee members for their great support, their time in reading this thesis, and their valuable suggestions.

I would like to thank Iyad A. Kanj, Ge Xia, Henry Brans, Songjian Lu and Fenghui Zhang for their helpful discussions and their collaboration on part of my research. Our collaboration work has been very helpful.

Many thanks go to all the teachers, friends and nice people in China and the US. They have given me many valuable and beautiful things in my life, which they themselves may not realize.

I would like to thank the National Science Foundation and the Department of Computer Science for the financial support.

Finally, I would like to thank my husband (who has been sharing all the happiness and hardship with me since we got married), our daughter (who is four year old as I finish this dissertation), and our whole family members: my grandparents, my parents, my parents-in-law, my brother and his wife, my sister-in-law and her husband and son, my aunts, my uncles, and my cousins.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

A. Motivation 1
B. Introduction to Parameterized Complexity Theory 6
C. Terminologies in Approximation 12
D. Thesis Outline 15

II PARAMETERIZED COMPLEXITY AND PTAS 17

A. Introduction 17
B. Efficient-FPT and FPTAS 18
C. Planar W-hierarchy and EPTAS 26
D. Remarks 38

III LOWER BOUNDS OF PARAMETERIZED COMPUTATION . 40

A. Parameterized NP-hard Problems 40
1. Introduction 40
2. Satisfiability and Weighted Satisfiability 43
3. Lower Bounds on Weighted Satisfiability Problems . . 52
4. Satisfiability Problems and the W-hierarchy 60
5. Linear fpt-reductions and Lower Bounds 67
B. On Some Parameterized Non NP-hard Problems 77
1. Further Remarks on $W_l[1]$-hardness 77
2. Parameterized LOGNP Problems 79

IV LOWER BOUNDS FOR PTAS ALGORITHMS 90

A. Our Theorem 90
B. The DSSP Problem 92
1. Standard Definitions of DSSP and Its PTAS 93
2. PTAS Lower Bound for DSSP 96
C. The LCS Problem 100
1. FLCS-k 102
2. LCS-λ 106
D. The LOGNP Problems 109

CHAPTER Page

1. Rich Hypergraph Cover, Tournament Dominating Set and V-C Dimension 109
2. LOG Hypergraph Cover, LOG Adjustment, and LOG Dominating Set 112
3. $\log n$-partite Graph Clique 117

V STUDY OF EPTAS ALGORITHMS ON PLANAR GRAPHS . 119

A. EPTAS Lower Bound Results 119
B. Planar Vertex Cover and EPTAS Upper Bound 120

VI CONCLUSIONS 131

A. Summary 131
B. Future Work 132

REFERENCES 136

VITA 145

LIST OF FIGURES

FIGURE		Page
1	An FPTAS algorithm for the problem Q.	22
2	Algorithm $A_{\geq}$.	91
3	Unfolding operation on the vertex v (with degree 6).	121
4	Parameterized algorithm for PLANAR VERTEX COVER.	125
5	EPTAS algorithm for PLANAR VERTEX COVER.	128

CHAPTER I

INTRODUCTION

A. Motivation

According to the NP-completeness theory, many problems that have important real-world applications are NP-hard [44]. There are no polynomial time algorithms for them unless P=NP. To deal with NP-hard problems, many approaches have been proposed. Approximation algorithms and parameterized computation are two of these approaches.

The highly acclaimed approximation approach [5] tries to come up with a *good* enough solution in polynomial time instead of an optimal solution for an NP-hard optimization problem. Several important approximation classes, which include FPTAS, EPTAS, and PTAS are introduced.

A notable class of NP-hard optimization problems has *fully polynomial-time approximation schemes* (FPTAS). An FPTAS algorithm is an efficient approximation algorithm whose approximation ratio is bounded by $1 + \epsilon$ and whose running time is bounded by a polynomial in both the input size and $1/\epsilon$, where the relative error bound ϵ can be any positive real number. Examples of FPTAS problems include the well-known KNAPSACK problem and the MAKESPAN problem on a fixed number of processors [50].

A more general class of NP-hard optimization problems admits *polynomial-time approximation schemes* (PTAS), which have polynomial time approximation algorithms of approximation ratio $1+\epsilon$ for each fixed relative error bound $\epsilon > 0$. A large number of NP-hard optimization problems belong to the class PTAS [50], including

---

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

the well-known EUCLIDEAN TRAVELING SALESMAN problem [4] and the GENERAL MULTIPROCESSOR JOB SCHEDULING problem [27]. Contrary to the efficiency of FPTAS algorithms, the running time of a general PTAS algorithm of approximation ratio $1+\epsilon$ can be of the form $O(n^{t(\epsilon)})$, where n is the input size and $t(\epsilon)$ is a function of ϵ that can be very large even for moderate values of ϵ. Downey [33] (see also Fellows [38]) examined many recently developed PTAS algorithms for NP-hard optimization problems, and discovered that for the relative error bound value of $\epsilon = 20\%$, most of these PTAS algorithms have $t(\epsilon) > 10^6$, i.e., the running time of these PTAS algorithms exceeds the order of n^{100000}! Obviously, these PTAS algorithms are not practically feasible.

Observing this fact, recent research has proposed to further refine the class PTAS. We say that an optimization problem has an *efficient polynomial-time approximation scheme* (EPTAS) if for any $\epsilon > 0$, there is an approximation algorithm of ratio $1+\epsilon$ whose running time is bounded by a polynomial of the input size whose degree is independent of ϵ. In particular, all FPTAS problems belong to the class EPTAS. EPTAS algorithms are superior to PTAS algorithms whose running time is of the form $O(n^{t(\epsilon)})$ in terms of the efficiency. In fact, many PTAS algorithms developed for NP-hard optimization problems are actually EPTAS algorithms. Moreover, there are a number of well-known NP-hard optimization problems, such as the EUCLIDEAN TRAVELING SALESMAN problem [4], the GENERAL MULTIPROCESSOR JOB SCHEDULING problem [27], and the MAKESPAN problem on unbounded number of processors [50], for which early developed PTAS algorithms had running time of the form $O(n^{t(\epsilon)})$, but later were improved to EPTAS algorithms.

The theory of parameterized complexity [37] is a newly developed approach introduced to address NP-hard problems with *small* parameters. It tries to give exact algorithms for an NP-hard problem when its natural parameter is small (even if the problem size is big). Problems are considered fixed-parameter tractable (in the class FPT) if they can be solved in time $O(f(k)n^c)$, where n is the problem size, k is the parameter, f is a recursive function, and c is a constant. For a problem in the class FPT, researchers try to come up with more efficient parameterized algorithms. For example, the VERTEX COVER problem is fixed-parameter tractable (in FPT).

> VERTEX COVER problem [20]: given a graph G and an integer k, determine if G has a vertex cover C of k vertices, i.e., a subset C of k vertices in G such that every edge in G has at least one end in C. Here the parameter is k.

The problem is a well-known NP-complete problem [44]. On the other hand, the Computational Biochemistry Research Group at the ETH Zürich has successfully applied algorithms for this problem to their research in multiple sequence alignments [73, 75], where the parameter value k can be bounded by 60. After many rounds of improvement, the best known algorithm for the VERTEX COVER problem runs in time $O(1.286^k + kn)$ [26]. This algorithm has been implemented and is quite practical [18].

Accompanying the work on designing efficient and practical parameterized algorithms, a theory of parameter intractability is developed. In parameterized complexity, to classify fixed-parameter intractable problems, a hierarchy, *the W-hierarchy* $\bigcup_{t\geq 0} W[t]$, where $W[t] \subseteq W[t+1]$ for all $t \geq 0$, has been introduced, in which the 0-th level $W[0]$ is the class *FPT*. The hardness and completeness have been defined for each level $W[i]$ of the W-hierarchy for $i \geq 1$, and a large number of $W[i]$-hard parameterized problems have been identified [37]. For example, the CLIQUE problem, the INDEPENDENT SET problem, and the DOMINATING SET problem are all $W[1]$-hard. Now it has become commonly accepted that no $W[1]$-hard (and $W[i]$-hard, $i > 1$) problem can be solved in time $f(k)n^{O(1)}$ for any function f (i.e., $W[1] \neq FPT$).

$W[1]$-hardness has served as the hypothesis for fixed-parameter intractability. Examples include a recent result by Papadimitriou and Yannakakis [67], showing that the DATABASE QUERY EVALUATION problem is $W[1]$-hard. This provides strong evidence that the problem cannot be solved by an algorithm whose running time is of the form $f(k)n^{O(1)}$, thus excluding the possibility of a practical algorithm for the problem even if the parameter k (the size of the query) is small as in most practical cases.

Also note that, as is pointed out in [20], "the theory of fixed-parameter tractability is not a simple refinement of the concept of NP-completeness", since there are fixed-parameter tractable problems which are harder than NP-complete problems, such as the ML TYPE-CHECKING problem, and there are also fixed-parameter intractable problems that seem easier than NP-complete problems, such as the V-C DIMENSION problem. "Therefore, the theory of fixed-parameter tractability seems to well supplement the theory of NP-completeness [20]."

Research activities in parameterized computation have demonstrated rich complexity structures and effective algorithmic approaches. This research area has found applications in computational biology, database systems, networks, parallel computing, VLSI design and other research areas. Please refer to [37, 33, 38, 20, 67, 47] and the recently published special issue in *Journal of Computer and System Sciences* (Volume 67, No.6, 2003, Guest Editors: J. Chen and M. Fellows).

We have seen that a lot of research has been done in the approximation area and the parameterized complexity area. The work on the connections of these two research areas is still primitive, but already demonstrates its beautiful theoretical properties and important practical applications. In the following are only a few nice results of the recent research work in this direction.

In [12], Cai and Chen proposed a standard approach to parameterize an NP optimization problem. Using the standard parameterization, they proved:

Lemma I.1 ([12]) *If an optimization problem has a fully polynomial-time approximation scheme, then the corresponding parameterized problem is fixed-parameter tractable (in FPT).*

Later this result was extended [17]:

Lemma I.2 ([17]) *All optimization problems that have efficient polynomial-time approximation schemes have their parameterized problems in FPT.*

This shows that for NP optimization problems whose corresponding parameterized problems are fixed-parameter intractable, they are unlikely to have efficient polynomial-time approximation schemes. The study of parameterized complexity "provides a new and potentially powerful approach to proving nonapproximability of NP optimization problems [37]."

As an application, Lemma I.2 was used to prove the lower bound result for the DISTINGUISHING SUBSTRING SELECTION problem (abbreviated DSSP) problem which arose in the area of computational biology. Gramm *et al.* in [46] proved that the DSSP problem is $W[1]$-hard. Combining this $W[1]$-hardness result with Lemma I.2, they got the following lower bound result for the problem:

Lemma I.3 ([46]) *Unless W[1]=FPT, the W[1]-hardness of DSSP excludes the possibility of DSSP having efficient polynomial-time approximation schemes.*

Therefore the PTAS algorithm for the DSSP problem designed in [30, 31] could not be greatly improved to an EPTAS algorithm.

In this thesis, we study the structures of parameterized problems with respect to their parameterized tractability and the relationship between parameterized complexity and approximability. Specifically, the work in this thesis includes the following:

- the study of the relationship between parameterized complexity and approximation classes.
- the investigation of the issues related to the computational lower bounds for NP-hard parameterized problems and some Non NP-hard parameterized problems.
- the extension of the techniques to derive computational lower bounds for PTAS and EPTAS approximation algorithms.

B. Introduction to Parameterized Complexity Theory

This section is adapted from some material in [20, 25]. Interested readers are referred to the book by Downey and Fellows [37] for a more systematic treatment of the theory of parameterized complexity. Here we only provide some fundamentals of parameterized complexity theory.

The theory of parameterized computation and complexity mainly considers decision problems (i.e., problems whose instances only require a yes/no answer). This losses no generality. In fact, it has been a very natural practice in the study of the NP-completeness theory [44] to reduce an optimization problem to a decision problem by introducing a parameter.

Definition A *parameterized problem* Q is a decision problem (i.e., a language) that is a subset of $\Sigma^* \times \mathcal{N}$, where Σ is a fixed alphabet and $\mathcal{N}$ is the set of all nonnegative integers. Thus, each element of Q is of the form (x, k), where the second component, i.e., the integer k, is the *parameter*.

A parameterized problem Q can take a more general form such that the parameter is also a finite string in a fixed alphabet [37]. Our discussion will be based on the above simplified definition in which the parameter is a nonnegative integer, as is the case for most parameterized problems.

We say that an algorithm A *solves* the parameterized problem Q if on each input (x, k), the algorithm A can determine whether (x, k) is a yes-instance of Q (i.e., whether (x, k) is an element of Q). We call the algorithm A a *parameterized algorithm* if its computational complexity is measured in terms of both the input length $|x|$ and the parameter value k.

Definition The parameterized problem Q is *fixed-parameter tractable* if it can be solved by a parameterized algorithm of running time bounded by $f(k)|x|^c$, where f is a recursive function and c is a constant independent of both k and $|x|$. Denote by FPT the class of all fixed-parameter tractable problems.

Many NP-hard parameterized problems, such as VERTEX COVER, are in the class FPT. For most developed parameterized algorithms for FPT problems, the recursive function f is moderate (e.g., $f(k) = d^k$ for a small constant $d > 1$). Therefore, for small parameter values of k, the running time $f(k)|x|^c$ of the algorithms for FPT problems becomes practically acceptable.

A natural question is whether there are parameterized problems (in particular, parameterized NP-complete problems) that are not fixed-parameter tractable. In order to discuss this, we first need to describe a group of satisfiability problems on circuits of bounded depth. For this, we first review some basic definitions and notations related to circuits.

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

Using the results in [19], 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$[t]$, consists of the pairs (C, k), where C is a Π_t-circuit and k is an integer, and 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 as WCS$[t]$ except that the circuit C is required to be monotone (resp. antimonotone). To simplify our discussion, we will denote by WCS$^*[t]$ the problem WCS$^+[t]$ when t is even, and the problem WCS$^-[t]$ when t is odd.

Finally, we define the problem WEIGHTED ANTIMONOTONE CNF 2-SAT (shortly

WCNF-2SAT$^-$) to be the set of pairs (F, k), where F is a CNF formula with only negative literals, in which each clause contains at most two literals and F has a satisfying assignment of weight k.

Extensive computational experience and practice have given strong evidences that the problem WCNF-2SAT$^-$ and the problems WCS$^*[t]$ for all $t > 1$ are not fixed-parameter tractable. The theory of fixed-parameter intractability is built based on this working hypothesis, which classifies the levels of fixed-parameter intractability in terms of the parameterized complexity of the problems WCNF-2SAT$^-$ and WCS$^*[t]$. For this, we need to introduce a new type of reduction.

Definition A parameterized problem Q is *fpt-reducible* to a parameterized problem Q' if there is an algorithm that transforms each instance (x, k) of Q into an instance (x', k') of Q' in time $O(f(k)|x|^c)$, where $k' = g(k)$, f and g are recursive functions and c is a constant, such that (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q'.

It is easy to verify that the fpt-reduction preserves the fixed-parameter tractability, in the following sense. Suppose that Q is fpt-reducible to Q'. Then if Q' is fixed-parameter tractable then so is Q, and if Q is not fixed-parameter tractable then neither is Q'.

Lemma I.4 ([37]) *Let Q_1, Q_2, and Q_3 be parameterized problems. If Q_1 is fpt-reducible to Q_2 and Q_2 is fpt-reducible to Q_3, then Q_1 is fpt-reducible to Q_3.*

Now we are ready to define the W-hierarchy [37].

Definition A parameterized problem Q_1 is in the class $W[1]$ if Q_1 is fpt-reducible to the problem WCNF-2SAT$^-$. A parameterized problem Q_t is in the class $W[t]$ for $t > 1$ if Q_t is fpt-reducible to the problem WCS$[t]$.

In particular, an *FPT* problem is in the class $W[t]$, for all $t \geq 1$. Moreover, observe that the WCNF-2SAT$^-$ problem is a subproblem of the WCS[2] problem and that the fpt-reduction is transitive, so we have $W[1] \subseteq W[2]$. By the similar reason, for an integer $t > 1$, since the problem WCS$[t]$ is trivially fpt-reducible to the problem WCS$[t+1]$, so $W[t] \subseteq W[t+1]$. Thus, if we define $W[0] = FPT$, then we obtain the fixed-parameter intractability hierarchy, the *W-hierarchy* $\bigcup_{t \geq 0} W[t]$, with

$$W[0] \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[t] \subseteq \cdots$$

In particular, by the definitions, the problem WCNF-2SAT$^-$ is in the class $W[1]$ and the problem WCS$[t]$ is in the class $W[t]$ for all $t > 1$. According to our working hypothesis, WCNF-2SAT$^-$ is not fixed-parameter tractable, which is equivalent to the statement $FPT \neq W[1]$.

Following the same style of NP-hardness and NP-completeness, we define:

Definition Let $t \geq 1$ be an integer. A parameterized problem Q_t is $W[t]$-*hard* if all problems in $W[t]$ are fpt-reducible to Q_t, and is $W[t]$-*complete* if in addition Q_t is also in $W[t]$.

By the definitions, we get a generic complete problem for each level in the W-hierarchy.

Theorem I.5 *The problem* WCNF-2SAT$^-$ *is* $W[1]$*-complete, and for all integers* $t > 1$*, the problem* WCS$[t]$ *is* $W[t]$*-complete.*

Since the fpt-reduction is transitive, we have

Theorem I.6 *For $t \geq 1$, if a $W[t]$-hard problem is fixed-parameter tractable, then $FPT = W[t]$.*

Since it is commonly believed that for all $t \geq 1$, $FPT \neq W[t]$, the $W[t]$-hardness of a parameterized problem provides a strong evidence that the problem is not fixed-parameter tractable.

The transitivity of the fpt-reduction also provides a convenient way for deriving hardness in the W-hierarchy.

Theorem I.7 *Let $t \geq 1$ be an integer. A parameterized problem Q_t is $W[t]$-hard if there is a $W[t]$-hard problem that is fpt-reducible to Q_t.*

In particular, for each integer $t \geq 2$, it can be shown that the problem WCS$[t]$ is fpt-reducible to the problem WCS$^*[t]$ [37]. The problem WCS$^*[t]$ is obviously in the class $W[t]$. Therefore, for each $t \geq 2$, we get the second $W[t]$-complete problem WCS$^*[t]$.

Using Theorem I.7, researchers in the theory of parameterized computation and complexity have identified over a hundred parameterized problems that are either hard or complete for various levels in the W-hierarchy [37]. For example, the problems INDEPENDENT SET, CLIQUE, and WEIGHTED 3-SAT are $W[1]$-complete, the problems WEIGHTED CNF-SAT, DOMINATING SET, SET COVER, HITTING SET, and 0-1 INTEGER PROGRAMMING are $W[2]$-complete. Many of these problems have been well-known for their theoretical and practical importance. Some of them have been the main targets for algorithmic research for many years. The fact that nobody has been able to develop a fixed-parameter tractable algorithm for any of these problems provides a strong support to our working hypothesis.

We point out that each level $W[t]$ of the W-hierarchy can also be defined in terms of the traditional machine models and of more "standard" complexity measures. See [13, 28, 40] for detailed discussions.

C. Terminologies in Approximation

For a reference of the theory of approximation, the readers are referred to the book [5]. In this section, we provide some basic terminologies for studying approximation algorithms and its relationship with parameterized complexity. These terminologies will be used through out this thesis.

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 $S_Q(x) = \{y : |y| \leq p(|x|) \text{ and } \pi(x, y)\}$;

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 $Q = (I_Q, S_Q, f_Q, opt_Q)$ if, for each input instance x in I_Q, 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:

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

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

The approximation algorithm A has an *approximation ratio* $r(n)$ 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 $|x|$.[1] Finally, an NP optimization problem Q has a *fully polynomial-time approximation scheme* (FPTAS) if it has a PTAS A_Q such that the running time of A_Q is bounded by a polynomial of $|x|$ and $1/\epsilon$.

Observe that the time complexity of a PTAS algorithm may be of the form $O(2^{1/\epsilon}|x|^c)$ for a fixed constant c or of the form $O(|x|^{1/\epsilon})$. Obviously, the latter type of computations with small ϵ values will turn out to be practically infeasible. This leads to the following definition [17].

Definition An NP optimization problem Q has an *efficient polynomial-time approximation scheme* (EPTAS) if it admits a polynomial-time approximation scheme whose time complexity is bounded by $O(f(1/\epsilon)|x|^c)$, where f is a recursive function

[1] There is an alternative definition for PTAS in which each $\epsilon > 0$ may correspond to a different approximation algorithm A_ϵ for Q [44]. The definition we adopt here may be called the *uniform PTAS*, by which a single approximation algorithm takes care of all values of ϵ. Note that most PTAS developed in the literature are uniform PTAS.

and c is a constant.

An NP optimization problem Q can be parameterized in a natural way as follows.

Definition 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 $Q_{\geq} = \{(x, k) \mid x \in I_Q \wedge opt_Q(x) \geq k\}$;

(2) If Q is a minimization problem, then the parameterized version of Q is defined as $Q_{\leq} = \{(x, k) \mid x \in I_Q \wedge opt_Q(x) \leq k\}$.

The above definition offers the possibility to study the relationship between the approximability and the parameterized complexity of NP optimization problems. However, there is an essential difference between the two categories: an approximation algorithm for an NP optimization problem constructs a solution for a given instance of the problem, while a parameterized algorithm only provides a "yes/no" decision on an input. To make the comparison meaningful, we need to extend the definition of parameterized algorithms in a natural way so that when a parameterized algorithm returns a "yes" decision, it also provides an "evidence" to support the conclusion (see [12] for a similar treatment).

Definition Let $Q = (I_Q, S_Q, f_Q, opt_Q)$ be an NP optimization problem. We say that *a parameterized algorithm A_Q solves the parameterized version of Q* if

(1) in case Q is a maximization problem, then on an input pair (x, k) in $Q_{\geq}$, the algorithm A_Q returns "yes" with a solution y in $S_Q(x)$ such that $f_Q(x, y) \geq k$, and on any input not in $Q_{\geq}$, the algorithm A_Q simply returns "no";

(2) in case Q is a minimization problem, then on an input pair (x, k) in $Q_{\leq}$, the algorithm A_Q returns "yes" with a solution y in $S_Q(x)$ such that $f_Q(x, y) \leq k$, and on any input not in $Q_{\leq}$, the algorithm A_Q simply returns "no".

D. Thesis Outline

The organization of this thesis is as follows. In Chapter II, we study the relationship between parameterized complexity and approximability. We present our characterizations of the approximation classes FPTAS and EPTAS using the theory of fixed-parameter tractability and the W-hierarchy of parameterized intractability.

In Chapter III, we study the structural properties of parameterized complexity, and introduce the definition of linear fpt-reduction. We investigate the issues related to the computational lower bounds for NP-hard parameterized problems, such as the INDEPENDENT SET, CLIQUE and dominating set problems, and some Non NP-hard parameterized problems, such as the problems in the class LOGNP.

In Chapter IV, we study the applications of parameterized complexity in deriving computational lower bounds on PTAS algorithms for NP-hard optimization problems, such as the DISTINGUISHING SUBSTRING SELECTION (DSSP) and LONGEST COMMON SUBSEQUENCE (LCS) problems, which have found important applications in computational biology. We then discuss the inapproximability of the problems in the class LOGNP.

In Chapter V, we derive computational lower bounds for EPTAS algorithms for some NP-hard problems on planar graphs. Since there is a gap between our lower bound results and the current upper bound results, in particular, we investigate the possibility of improving the upper bound of the EPTAS algorithm for the planar vertex cover problem.

We give a summary of our work and the directions for future research in Chapter VI.

CHAPTER II

PARAMETERIZED COMPLEXITY AND PTAS*

This chapter is joint work with J. Chen, I. Kanj, and G. Xia [24].

A. Introduction

In this chapter, we study the relationship between the approximability and the parameterized complexity of NP optimization problems.

We start by identifying a subclass, *efficient-FPT*, of fixed parameter tractable problems, and prove that under a very general condition (the *scalability* condition, see the next section for a formal definition), a problem is in FPTAS if and only if it is in efficient-FPT. This provides a very precise characterization of the approximation class FPTAS in terms of parameterized complexity. This characterization has advantages over the previous characterizations for the class FPTAS. Compared to Paz and Moran's characterization of the class FPTAS based on certain polynomial time computable functions [68] (see also [6]), our characterization is easier to verify: the scalability condition seems to be satisfied by almost all NP optimization problems. Compared to Woeginger's recent characterization of the class FPTAS based on a dynamic programming formulation, our characterization seems more general and includes more FPTAS problems.

We then study the characterization of the class EPTAS. We enforce a constraint of planarity on the W-hierarchy in parameterized complexity theory, and introduce

---

*Part of the data reported in this chapter is reprinted with permission from "Polynomial time approximation schemes and parameterized complexity" by J. Chen, X. Huang, I. Kanj, and G. Xia, 2004, *Proceedings of the 29th International Symposium on the Mathematical Foundations of Computer Science*, (MFCS 2004), pp. 500-512, the syntactic classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT (this approach is similar to that of Khanna and Motwani [57] in their efforts to characterize the class PTAS). These syntactic classes capture many NP optimization problems in the class EPTAS, such as PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and PLANAR MAX-SAT. By extending Baker's techniques [7] and techniques more recently developed in the study of parameterized algorithms [2, 43], we prove that all problems in these syntactic classes belong to the class EPTAS. These syntactic classes seem to form the core for a significant class of EPTAS problems. Finally, we point out that our syntactic classes are significantly different from the PTAS syntactic classes introduced by Khanna and Motwani [57]: our syntactic classes characterize only EPTAS problems while the syntactic classes in [57] seem to include PTAS problems that are not in EPTAS, while on the other hand, our syntactic classes contain EPTAS problems that cannot be characterized by the syntactic classes in [57].

Our results combined with a result by Cesati and Trevisan [17] show that all problems expressible by our syntactic classes are fixed-parameter tractable. Moreover, a byproduct derived from an immediate result in our discussion shows that for any fixed integer $t \geq 0$, the PLANAR t-NORMALIZED WEIGHTED SATISFIABILITY problem is solvable in polynomial time, which answers an open problem posed by Downey and Fellows [37].

B. Efficient-FPT and FPTAS

In this section, we present a characterization for the approximation class FPTAS in terms of parameterized complexity. Recall that a fixed-parameter tractable problem has an algorithm of running time of the form $f(k)n^c$, where f is an arbitrary recursive function. By enforcing a further constraint on the function $f(k)$, we introduce the following subclass of the class FPT:

Definition An NP optimization problem Q is *efficiently fixed-parameter tractable* (efficient-FPT) if its parameterized version is solvable by a parameterized algorithm of running time bounded by a polynomial of $|x|$ and k.

Note that efficient-FPT does not necessarily imply polynomial time computability: NP optimization problems, in particular a large variety of scheduling problems, may have their optimal values much larger than the input size. In consequence, the parameterized versions of these problems may have their parameter values k much larger than the input size.

Definition An optimization problem $Q = (I_Q, S_Q, f_Q, opt_Q)$ is said to be *scalable* if there are polynomial time computable functions g_1 and g_2 and a fixed polynomial q such that:

1. for any instance $x \in I_Q$, and any integer $d \geq 1$, $x_d = g_1(x, d)$ is an instance of Q such that $|x_d| \leq q(|x|)$ and $|opt_Q(x_d) - opt_Q(x)/d| \leq q(|x|)$; and

2. for any solution y_d to the instance x_d, $y = g_2(x_d, y_d)$ is a solution to the instance x such that $|f_Q(x_d, y_d) - f_Q(x, y)/d| \leq q(|x|)$.

Most NP optimization problems are scalable. In particular, if an NP optimization problem Q has its optimal value $opt(x)$ bounded by a polynomial of $|x|$ for all instances x, then the problem Q is automatically scalable — simply let $x_d = g_1(x, d) = x$ for any integer d, and for a solution y_d to $x_d = x$, let $g_2(x_d, y_d) = y_d$. This immediately implies that most set problems and graph problems are scalable, including the well-known NP-hard problems such as BIN PACKING, 3D-MATCHING, SET COVER, VERTEX

COVER, and DOMINATING SET. Moreover, most NP optimization problems involving large numbers (i.e., the *number problems* defined by Garey and Johnson [44]), such as KNAPSACK and MAKESPAN, are also scalable. We pick Q = MAKESPAN as an example to illustrate how such a problem involving large numbers can be scaled. An instance x of MAKESPAN consists of n jobs of integral processing times t_1, t_2, ..., t_n, respectively (we will refer to the jth job by t_j), and an integer m, the number of identical processors, and asks to construct a scheduling of the jobs on the m processors so that the completion time (i.e., the *makespan*) is minimized. For a given instance $x = (t_1, t_2, \ldots, t_n; m)$ of MAKESPAN and a given integer $d \geq 0$, we define

$$x_d = g_1(x, d) = (t'_1, t'_2, \ldots, t'_n; m)$$

where $t'_i = \lceil t_i/d \rceil$ for $i = 1, 2, \ldots, n$, which is also an instance for MAKESPAN. A solution y_d to the instance x_d is a scheduling that partitions the n jobs in x_d into m subsets: $y_d = (T'_1, \ldots, T'_m)$, where T'_i is the set of jobs in x_d that are assigned to the ith processor. We define $y = g_2(x_d, y_d)$ to be the same index partitioning of the jobs in x: $y = (T_1, \ldots, T_m)$ (i.e., a job t_j is in T_i if and only if the job t'_j is in T'_i). Obviously, $y = g_2(x_d, y_d)$ is a solution for the instance x, and the functions g_1 and g_2 are computable in polynomial time. To see the relation between the solution $y = (T_1, \ldots, T_m)$ for x and the solution $y_d = (T'_1, \ldots, T'_m)$ for x_d, note that the makespan of y is equal to $\max_i\{\sum_{t_j \in T_i} t_j\}$, and the makespan of y_d is equal to $\max_i\{\sum_{t'_j \in T'_i} t'_j\}$. We have

$$\begin{aligned} f_Q(x_d, y_d) &= \max_i\{\sum_{t'_j \in T'_i} t'_j\} = \max_i\{\sum_{t_j \in T_i} \lceil t_j/d \rceil\} \\ &\geq \max_i\{\sum_{t_j \in T_i} t_j/d\} = \max_i\{\sum_{t_j \in T_i} t_j\}/d = f_Q(x, y)/d. \end{aligned} \tag{2.1}$$

On the other hand

$$\begin{aligned} f_Q(x_d, y_d) &= \max_i \{ \sum_{t'_j \in T'_i} t'_j \} = \max_i \{ \sum_{t_j \in T_i} \lceil t_j/d \rceil \} \\ &\leq \max_i \{ \sum_{t_j \in T_i} (t_j/d + 1) \} \leq \max_i \{ \sum_{t_j \in T_i} t_j \}/d + n \\ &= f_Q(x, y)/d + n. \end{aligned} \tag{2.2}$$

Here we have used the fact that the total number of jobs in each subset T_i is bounded by n. Combining (2.1) and (2.2), we get $|f_Q(x_d, y_d) - f_Q(x, y)/d| \leq n$. Similarly, it can be verified that the instances x and x_d satisfy $|opt_Q(x_d) - opt_Q(x)/d| \leq n$. In conclusion, the MAKESPAN problem is scalable.

Theorem II.1 *Let $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ be a scalable NP optimization problem. Then Q has an FPTAS if and only if Q is efficient-FPT.*

PROOF. One direction of the theorem was implicitly proved in [12]. Suppose that Q has an FPTAS A_Q, which is an algorithm such that on any instance x of Q and any given $\epsilon > 0$, the algorithm A_Q constructs a solution of ratio bounded by $1 + \epsilon$ for x, in time $p(|x|, 1/\epsilon)$, where $p(|x|, 1/\epsilon)$ is a polynomial of $|x|$ and $1/\epsilon$. Cai and Chen proved ([12], Theorem 3.2) that then the parameterized version of Q can be solved in time $O(p(|x|, 2k))$. In consequence, the problem Q is efficient-FPT.

To show the converse, we consider specifically the case when Q is a maximization problem (a proof for minimization problems can be similarly derived). Suppose that the problem Q is efficient-FPT, and the parameterized version $Q_{\geq}$ is solvable in time $p(k, |x|)$, which is a polynomial in k and $|x|$. Since Q is scalable, we let g_1 and g_2 be the polynomial time computable functions, and q be the polynomial in the definition of the scalability of Q. For a given instance x of Q and a real number $\epsilon > 0$, consider the algorithm (assume $n = |x|$) shown in Fig 1.

FPTAS Algorithm for Q.

begin

1. let $x_1 = g_1(x, 1)$; if $(x_1, 3q(n)/\epsilon)$ is not in $Q_{\geq}$, then try all instances $(x, 1)$, $(x, 2)$, $\ldots$, $(x, 3q(n)/\epsilon + q(n))$ to construct an optimal solution for x; STOP.

2. use binary search on d to find an integer $d \geq 1$ such that $(x_d, 3q(n)/\epsilon)$ is in $Q_{\geq}$, but $(x_{d+1}, 3q(n)/\epsilon)$ is not in $Q_{\geq}$;

3. construct an optimal solution y_d for the instance x_d;

4. let $y_0 = g_2(x_d, y_d)$; output y_0 as a solution for x.

end

Fig. 1. An FPTAS algorithm for the problem Q.

We discuss the correctness and the complexity of the above algorithm. First note that by the definition, $|x_d| \leq q(n)$ for any integer d. If $(x_1, 3q(n)/\epsilon)$ is not in $Q_{\geq}$, then $opt_Q(x_1) < 3q(n)/\epsilon$. Moreover, since Q is scalable, we have $|opt_Q(x_1) - opt_Q(x)/1| \leq q(n)$. Combining these two relations, we get $opt_Q(x) \leq opt_Q(x_1) + q(n) < 3q(n)/\epsilon + q(n)$. Thus, step 1 of the algorithm will correctly construct an optimal solution for the instance x (note by our definition, on input $(x, opt_Q(x))$, the parameterized algorithm must return "yes" with an optimal solution to the instance x). Moreover, since checking each instance (x, k) takes time $p(k, n)$, where $k = 1, 2, \ldots, 3q(n)/\epsilon + q(n)$, step 1 of the algorithm takes time bounded by $O((3q(n)/\epsilon + q(n))p(3q(n)/\epsilon + q(n), n))$, which is a polynomial of n and $1/\epsilon$.

If $(x_1, 3q(n)/\epsilon)$ is in $Q_{\geq}$, then we execute step 2 of the algorithm. First we need to show that there must be an integer $d \geq 1$ such that $(x_d, 3q(n)/\epsilon)$ is in $Q_{\geq}$ but

$(x_{d+1}, 3q(n)/\epsilon)$ is not in $Q_{\geq}$. We already know that $(x_d, 3q(n)/\epsilon)$ is in $Q_{\geq}$ for $d = 1$. Thus, we only need to show that there must be a d such that $(x_d, 3q(n)/\epsilon)$ is not in $Q_{\geq}$. Since Q is an NP optimization problem, we have $opt_Q(x) < 2^{r(n)}$, where $r(n)$ is a polynomial in n. Therefore if we let $d = 2^{r(n)}$, then from the scalability of the problem Q, we have $|opt_Q(x_d) - opt_Q(x)/d| \leq q(n)$, which gives immediately $opt_Q(x_d) < 1+q(n) \leq 3q(n)/\epsilon$ (here we assume without loss of generality that $q(n) \geq 1$ and $0 < \epsilon < 1$). Thus, the integer d in step 2 of the algorithm must exist and $d \leq 2^{r(n)}$. Since we use binary search on d, the total number of instances $(x_d, 3q(n)/\epsilon)$ we check in step 2 is bounded by $r(n)$. By our assumption, each instance $(x_d, 3q(n)/\epsilon)$ of $Q_{\geq}$ can be tested in time $p(3q(n)/\epsilon, q(n))$ (note that $|x_d| \leq q(n)$). Therefore, the running time of step 2 of the algorithm is also bounded by a polynomial of n and $1/\epsilon$.

Now consider step 3. Since $(x_{d+1}, 3q(n)/\epsilon)$ is not in $Q_{\geq}$, we have $opt_Q(x_{d+1}) < 3q(n)/\epsilon$. By the scalability of Q, we have

$$|opt_Q(x_d) - opt_Q(x)/d| \leq q(n)$$
$$|opt_Q(x_{d+1}) - opt_Q(x)/(d+1)| \leq q(n)$$

From this we get (note since $d \geq 1$, we have $(d+1)/d \leq 2$)

$$\begin{aligned}
opt_Q(x_d) &\leq \frac{opt_Q(x)}{d} + q(n) \\
&= \frac{d+1}{d} \cdot \frac{opt_Q(x)}{d+1} + q(n) \\
&\leq \frac{d+1}{d}(opt_Q(x_{d+1}) + q(n)) + q(n) \\
&\leq 2 \cdot opt_Q(x_{d+1}) + 3q(n) \\
&\leq \frac{6q(n)}{\epsilon} + 3q(n)
\end{aligned}$$

Thus, by checking all instances (x_d, k), where $k = 1, 2, \ldots, 6q(n)/\epsilon + 3q(n)$, each taking time $p(k, q(n))$, we will be able to construct the optimal solution y_d for the instance x_d. In conclusion, step 3 of the algorithm also takes time polynomial in n and $1/\epsilon$.

Summarizing the above discussion, we conclude that the running time of the algorithm is bounded by a polynomial in n and $1/\epsilon$. What remains is to bound the approximation ratio for the solution y_0 of the instance x.

By our construction, $f_Q(x_d, y_d) = opt_Q(x_d)$ and $y_0 = g_2(x_d, y_d)$. By the scalability of Q,

$$|opt_Q(x_d) - f_Q(x, y_0)/d| = |f_Q(x_d, y_d) - f_Q(x, y_0)/d| \leq q(n) \tag{2.3}$$

Thus, $f_Q(x, y_0) \geq d \cdot opt_Q(x_d) - d \cdot q(n)$. Since $(x_d, 3q(n)/\epsilon)$ is in $Q_{\geq}$, we have $opt_Q(x_d) \geq 3q(n)/\epsilon$, which gives (note $0 < \epsilon < 1$ thus $d/\epsilon \geq d$):

$$f_Q(x, y_0) \geq 3d \cdot q(n)/\epsilon - d \cdot q(n) = q(n)(3d/\epsilon - d) \geq 2dq(n)/\epsilon \tag{2.4}$$

Now from (2.3) and the inequality $|opt_Q(x_d) - opt_Q(x)/d| \leq q(n)$, we get

$$|opt_Q(x)/d - f_Q(x, y_0)/d| \leq |opt_Q(x_d) - opt_Q(x)/d| + |opt_Q(x_d) - f_Q(x, y_0)/d| \leq 2q(n)$$

Thus $opt_Q(x) - f_Q(x, y_0) \leq 2dq(n)$ (recall that Q is a maximization problem). This eventually gives us

$$opt_Q(x)/f_Q(x, y_0) \leq 1 + 2dq(n)/f_Q(x, y_0) \leq 1 + \epsilon$$

The last inequality is from (2.4). In conclusion, the approximation ratio of the solution y_0 for the instance x is bounded by $1 + \epsilon$.

This proves that the algorithm above is an FPTAS for the problem Q. This completes the proof of the theorem. □

As an application of Theorem II.1, the scalability as shown earlier and the well-known dynamic programming algorithm of running time $O(nk^m)$ [44] for the

MAKESPAN problem conclude immediately that the MAKESPAN problem has an FPTAS when the number m of processors is a fixed constant. This is a major result in [74].

We make a few remarks on Theorem II.1. Since the first group of publications on FPTAS for NP optimization problems [52, 74], there has been a line of research trying to characterize problems in FPTAS [6, 68, 79]. Most of the early work in this direction [6, 68] characterizes the class FPTAS in terms of certain polynomial time computable functions. These characterizations do not provide any clue on how to detect the existence of such functions, or on how to develop FPTAS for the problems (the interested readers are referred to [68], Theorem 4.20, for a more detailed discussion on this line of research). Very recently, Woeginger [79], in an effort to overcome this difficulty, considered a class of optimization problems that can be formulated via dynamic programming of certain structures. He showed that as long as the cost and transition functions of such problems satisfy certain arithmetical and structural conditions, the problems have FPTAS.

In comparison to these related works, Theorem II.1 seems to have the following advantages. First, as we have shown for the MAKESPAN problem, the scalability property of an NP optimization problem is satisfied in most cases and, in general, can be checked in a straightforward manner. Thus, in most cases, the existence of FPTAS for an NP optimization problem is reduced to the development of an efficient-FPT algorithm for the problem. Moreover, the proof of Theorem II.1 describes in detail how an efficient-FPT algorithm is converted into an FPTAS algorithm. On the other hand, Theorem II.1 seems to cover more FPTAS problems than Woeginger's characterization [79]: intuitively, and generally, a dynamic programming formulation for an NP optimization problem directly implies an efficient-FPT algorithm for the problem.

C. Planar W-hierarchy and EPTAS

In the previous section, we have shown how a subclass of the parameterized class FPT, the class efficient-FPT, provides a nice characterization for the approximation class FPTAS. In this section, we study the approximation class EPTAS in terms of the parameterized class, the W-hierarchy.

We note that a significant amount of research has been done on studying the approximation properties in terms of their syntactic descriptions. For instance, Papadimitriou and Yannakakis [65] introduced the syntactic classes MAXNP and MAXSNP of optimization problems, which, via proper approximation ratio preserving reductions, turn out to be exactly the class of NP optimization problems that can be approximated in polynomial time with constant approximation ratios [58]. Khanna and Motwani [57] proposed the syntactic classes MPSAT, TMAX, and TMIN by enforcing a planar structure on first order Boolean formulas of depth 3, and showed that most known PTAS problems are expressible by these classes.

In a parallel approach to that of Khanna and Motwani [57], we study the approximation class EPTAS by enforcing a planar structure on the W-hierarchy in parameterized complexity. A Π_h-circuit is a Π_h^+-*circuit* if all of its inputs are labeled by positive literals, and is a Π_h^--*circuit* if all of its inputs are labeled by negative literals. A Π_h-circuit α is *planar* if α becomes a planar graph after removing the output gate in α.

Definition We define the following syntactic optimization classes:

PLANAR MIN-$W[h]$: consists of every optimization problem Q such that each instance of Q can be expressed as a planar Π_h^+-circuit α, and the problem is to look for a satisfying assignment of minimum weight for α.

PLANAR MAX-$W[h]$: consists of every optimization problem Q such that each instance of Q can be expressed as a planar Π_h^--circuit α, and the problem is to look for a satisfying assignment of maximum weight for α.

PLANAR $W[h]$-SAT: consists of every optimization problem Q such that each instance of Q can be expressed as a planar Π_h-circuit α, and the problem is to look for an assignment that satisfies the largest number of depth-$(h-1)$ gates in the circuit α.

We make a few remarks on the above optimization classes. The classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT are optimization versions, with a planarity constraint, of the problem WCS(h), which is the representative complete problem for the hth level $W[h]$ of the W-hierarchy in parameterized complexity theory. The class PLANAR $W[h]$-SAT captures the optimization problems such as the PLANAR MAXIMUM SATISFIABILITY problem, where the objective is to construct a solution that satisfies the maximum number of constraints. In particular, the problem PLANAR MAXSAT formulated by Khanna and Motwani [57] belongs to the class PLANAR $W[2]$-SAT. The classes PLANAR MIN-$W[h]$ and PLANAR MAX-$W[h]$ capture the optimization problems where the objective is to construct an optimal (minimum or maximum) solution that satisfies all the constraints. Most optimization problems on planar graphs belong to the classes PLANAR MIN-$W[h]$ or PLANAR MAX-$W[h]$. For example, for an instance G of the MINIMUM VERTEX COVER on planar graphs, we can convert G into a planar Π_2^+-circuit α_G by making each vertex v in G an input of α_G and replacing each edge $[v, w]$ in G by an OR gate with the two inputs v and w, which is connected to the unique output AND gate of the circuit α_G. It is easy to see that the minimum vertex covers of the graph G correspond to the minimum weight assignments that satisfy the circuit α_G, and vice versa.

In the rest of this section, we show that all optimization problems expressible by our syntactic classes have EPTAS. EPTAS algorithms for these problems are developed based on methods similar to that presented in [7]. We provide the details below, emphasizing on the differences. Moreover, since the algorithms for the three classes are similar, we will concentrate on the class PLANAR MIN-$W[h]$, and give brief explanations on how the algorithms can be modified to apply to the classes PLANAR MAX-$W[h]$ and PLANAR $W[h]$-SAT.

Let G be a planar graph (not necessarily connected) and $\pi(G)$ be a planar embedding of G. A vertex v is in *layer*-1 in $\pi(G)$ if v is on the boundary of the unbounded region of $\pi(G)$. We define G_1 to be the subgraph of G induced by all layer-1 vertices. Inductively, a vertex v is in layer-i, $i > 1$, if v is on the unbounded region of the embedding of the graph $G-(G_1 \cup \ldots \cup G_{i-1})$ induced by the embedding $\pi(G)$. Define G_i to be the subgraph of G induced by all layer-i vertices. The embedding $\pi(G)$ is *q-outerplanar* if it has at most q layers.

Now consider a planar Π_h^+-circuit α_w with output gate w. Let $\alpha = \alpha_w - w$ be the subgraph of α_w with the output gate w removed. By the definition, the graph α has a planar embedding $\pi(\alpha)$. Let G be a subgraph of α that is induced by q consecutive layers in $\pi(\alpha)$, where $q \geq 2h$, and let $\pi(G)$ be the embedding of G induced from $\pi(\alpha)$. Obviously, the embedding $\pi(G)$ is q-outerplanar. We consider the following optimization problem:

> MIN (h, q)-SAT
>
> Given the graph G and the q-outerplanar embedding $\pi(G)$ of G, as defined above, construct an assignment of minimum weight for the input variables in G so that all depth-$(h-1)$ gates in α_w that are in the middle $q-2h$ layers in $\pi(G)$ (i.e., the $(h+1)$st, ..., and the $(q-h)$th layers in $\pi(G)$) are satisfied.

We point out that assigning all input variables in G the value 1 will satisfy all depth-$(h-1)$ gates in the middle $q-2h$ layers in $\pi(G)$. This is because all literals in α_w are positive, and α_w has depth h. So any input variable or any gate that is connected via a path in α_w to a depth-$(h-1)$ gate in the middle $q-2h$ layers in $\pi(G)$ must necessarily be contained in G, and hence, when all these input variables in G are assigned the value 1, all the depth-$(h-1)$ gates in the middle $q-2h$ layers in $\pi(G)$ will be satisfied.

Lemma II.2 *The problem* MIN (h,q)-SAT *can be solved in time* $O(81^q n)$*.*

PROOF. The proof proceeds based on the techniques proposed by Baker [7]. Starting with the q-outerplanar embedding $\pi(G)$, we can recursively decompose the graph G into "slices". Each slice S is a subgraph of G with at most q "left boundary vertices" and at most q "right boundary vertices", which are the only vertices in S that may be adjacent to vertices not in S. A *trivial slice* is simply an edge in G. Two slices S_1 and S_2 can be "merged" into a larger slice S if the right boundary of S_1 is identical to the left boundary of S_2. Baker [7] presented a linear time algorithm to show how a q-outerplanar graph G is decomposed into slices and how the slices, starting from trivial slices, are recursively merged to reconstruct the original graph G.

To use the slice decomposition of the graph G to solve the MIN (h,q)-SAT problem, we assign a value to each boundary vertex v in a slice S in G. The boundary vertex v may have the following possible values (note that all inputs of an OR gate are either an input variable or an AND gate, and all inputs of an AND gate are either an input variable or an OR gate):

- If v is an input variable, then v may have value either 0 or 1.
- If v is an OR gate, then v may have three possible values:

(1) value 0, in this case all inputs of v in S should have value either 0 or $\tilde{0}$;

(2) value 1, if v has value 1 and an input of v in S has value 1; or

(3) value $\tilde{1}$, if v has value 1 but no input of v in S has value 1.

- If v is an AND gate, then v may have three possible values:

(1) value 1, in this case all inputs of v in S should have value either 1 or $\tilde{1}$;

(2) value 0, if v has value 0 and an input of v in S has value 0; or

(3) value $\tilde{0}$, if v has value 0 but no input of v in S has value 0.

We call a possible value assignment to the vertices in a (either left or right) boundary of a slice a "configuration" of the boundary. Each slice S with left boundary L and right boundary R is associated with a "table" T_S. For each configuration f_L of L and each configuration f_R of R, the table T_S records a minimum weight assignment $A_{\min}(S, f_L, f_R)$ to the input variables in the slice S that realizes the configurations f_L and f_R on the boundaries L and R, and satisfies all depth-$(h-1)$ gates that are in S and belong to the middle $q - 2h$ layers in $\pi(G)$. Since each vertex in L and R may have at most three different values and the total number of vertices in $L \cup R$ is bounded by $2q$, the table T_S has at most 3^{2q} items. If S is simply a trivial slice, then the table T_S can be constructed by enumerating all possible situations.

To recursively construct the tables for larger slices, we need to merge two slices S_1 and S_2 into a larger slice S. Suppose that the left and right boundaries of S_1 and S_2 are L_1 and R_1, and L_2 and R_2, respectively. The left and right boundaries of the larger slice S will be L_1 and R_2. By the construction described in [7], the right boundary R_1 of S_1 is identical to the left boundary L_2 of S_2. Now fix a configuration f_{L_1} of L_1 and a configuration f_{R_2} of R_2. By enumerating all pairs of consistent configurations f_{R_1} of R_1 and f_{L_2} of L_2, and by reading the records $A_{\min}(S_1, f_{L_1}, f_{R_1})$ and $A_{\min}(S_2, f_{L_2}, f_{R_2})$ in the tables T_{S_1} and T_{S_2}, we will be able to construct the assignment $A_{\min}(S, f_{L_1}, f_{R_2})$

for the larger slice S.

We explain what we mean by "a pair of consistent configurations" f_{R_1} and f_{L_2} of the boundaries R_1 and L_2. Let v be a vertex on the boundaries $R_1 = L_2$. If v is an input variable, then the value assignments on v are consistent if f_{R_1} and f_{L_2} assign the same value, 0 or 1, to v. If v is an OR gate, then the value assignments on v are consistent if either (1) both f_{R_1} and f_{L_2} assign the same value to v; or (2) one of f_{R_1} and f_{L_2} assigns the value 1 and the other assigns the value $\tilde{1}$ to v; or (3) the vertex v is also on the boundaries $L_1 \cup R_2$ and both f_{R_1} and f_{L_2} assign value $\tilde{1}$ to v (in this case, the vertex v will have value $\tilde{1}$ on the boundaries of the larger slice S). The case that v is an AND gate can be similarly described. Finally, the configurations f_{R_1} and f_{L_2} are consistent if their value assignments to every vertex in $R_1 = L_2$ are consistent.

Since each boundary vertex v may have three possible values, and each (left or right) boundary has at most q vertices, for each fixed pair of configurations f_{L_1} and f_{R_2} of the boundaries L_1 and R_2, there are at most $3^q \cdot 3^q$ possible pairs of configurations on the boundaries R_1 and L_2. Thus, the record $A_{\min}(S, f_{L_1}, f_{R_2})$ can be constructed in time $O(9^q)$. In consequence, the table T_S, which has a record for each pair of configurations f_{L_1} and f_{R_2} of the boundaries L_1 and R_2, can be constructed in time $O(9^q \cdot 9^q) = O(81^q)$.

Using Baker's algorithm which recursively decomposes the q-outerplanar graph G into slices and reconstructs the graph G from its trivial slices by recursively merging slices, we conclude that in time $O(81^q n)$,[1] we can construct a minimum weight assignment to the input variables in G that satisfies all depth-$(h-1)$ gates that are in the middle $q - 2h$ layers in G, thus solving the MIN (h, q)-SAT problem. □

[1]We remark that based on the approach of graph tree decomposition and more careful slice merging [2], the complexity of the algorithm described in Lemma II.2 can be improved to $O(c^q n)$ for a constant c much smaller than 81. However, this will not affect our main results.

Downey and Fellows ([37], page 482) posed an open problem for the parameterized complexity of the following problem:

PLANAR t-NORMALIZED WEIGHTED SATISFIABILITY

Given a π_t-circuit α that is a planar graph in the strict sense (i.e., it is planar even without removing the output gate) and a parameter k, does α have a satisfying assignment of weight k?

Fix a planar embedding $\pi(\alpha)$ of the circuit α. Suppose the output gate of α is contained in the ith layer L_i in $\pi(\alpha)$. Since the depth of α is bounded by t, every gate in α must be contained in one of the $2t+1$ layers L_{i-t}, …, L_i, …, L_{i+t}. In consequence, the embedding $\pi(\alpha)$ must be $(2t+1)$-outerplanar. Thus, similar to the proof of Lemma II.2, we can construct a satisfying assignment of weight k to the circuit α based on the slice structure of $\pi(\alpha)$ (or report no such an assignment exists). The only difference is that now for each boundary configuration (f_L, f_R) of a slice S, we should record *all* possible weights w, $0 \leq w \leq k$, such that there is an assignment to the input variables in the slice S that implements the boundary configuration (f_L, f_R). Now merging two slices should also consider combining all possible weights recorded in the two slices, which increases the time complexity by a factor of $O((2t+1)^2)$. Therefore, this induces an algorithm of running time $O(81^{2t+1}(t+1)^2 n)$ for solving the problem PLANAR t-NORMALIZED WEIGHTED SATISFIABILITY.

Theorem II.3 *For each integer t, the* PLANAR t-NORMALIZED WEIGHTED SATISFIABILITY *is solvable in polynomial time.*

Now we return back to the discussion of the problem PLANAR MIN-$W[h]$.

Theorem II.4 *For every $h \geq 1$,* PLANAR MIN-$W[h]$ *is a subclass of the class EPTAS.*

Proof. We present an EPTAS algorithm for a given planar min-$W[h]$ problem Q.

For a given constant $\epsilon > 0$ and an instance G_w of the problem Q, where G_w is a planar Π_h^+-circuit with output gate w, we first construct a planar embedding $\pi(G)$ for the graph $G = G_w - w$, and let $q = 2h(\lceil 1/\epsilon \rceil + 1)$. By adding empty layers to the embedding $\pi(G)$, we can assume without loss of generality that the layers of the embedding $\pi(G)$ are $L_1, L_2, \cdots, L_r$, where $r > 2q + 2h$, and the first $q + h$ layers L_1, $\ldots$, L_{q+h}, and the last $q + h$ layers $L_{r-q-h+1}$, $\ldots$, L_r are all empty.

For each fixed integer d, where $0 \leq d \leq q/(2h) - 2$, we construct a decomposition D_d of overlapping "chunks" of the graph G. Each chunk consists of q consecutive layers of the embedding $\pi(G)$, and two overlapping chunks share $2h$ common layers. More formally, for $i \geq 0$, the ith chunk of the decomposition D_d consists of the q layers L_j, where $2hd + i(q - 2h) + 1 \leq j \leq 2hd + i(q - 2h) + q$, and i satisfies $2hd + i(q - 2h) + q \leq r$. By our assumption, the layers that do not belong to any chunk in D_d are all empty layers, and the first h layers in the 0th chunk in D_d, and the last h layers in the last chunk in D_d are also empty layers.

Let U_i be the ith chunk of G and $\pi(U_i)$ be the q-outerplanar embedding of U_i induced from the embedding $\pi(G)$. According to Lemma II.2, in time $O(81^q n_i)$ we can construct a minimum weight assignment f_{d,U_i} to the input variables in U_i that satisfies all depth-$(h-1)$ gates in the middle $q - 2h$ layers in $\pi(U_i)$, where n_i is the total number of vertices in U_i.

Now merge the assignments f_{d,U_i} over all chunks U_i of D_d to obtain an assignment f_d for the input variables in G (i.e., if v belongs to a single chunk U_i in G, then $f_d(v) = f_{d,U_i}(v)$, while if v is shared by two consecutive chucks U_i and U_{i+1} in G, then $f_d(v) = f_{d,U_i}(v) \vee f_{d,U_{i+1}}(v)$). Since two consecutive chunks overlap with $2h$ layers, every depth-$(h-1)$ gate in G belongs to the middle $q-2h$ layers for some chuck. Moreover, the circuit G_w is monotone in the sense that if an assignment f satisfies a gate v then changing any 0 bit in f into 1 also makes an assignment satisfying the gate v. Therefore, the assignment f_d to the input variables in G satisfies all depth-$(h-1)$ gates in G, thus satisfies the circuit G_w. It is easy to see from the above discussion that the assignment f_d can be constructed in time $O(81^q n)$, where n is the total number of vertices in G.

For each integer d, $0 \leq d \leq q/(2h)-2$, we construct the assignment f_d to the input variables in G_w that satisfies the circuit G_w. We pick the one f_d with minimum weight over all d and output it as our solution f_{apx}. Let the weight of f_{apx} be $|f_{apx}|$.

Thus, in time $O(81^q n) = O(81^{2h/\epsilon} n)$, the above algorithm constructs an assignment f_{apx} that satisfies the given planar Π_h^+-circuit G_w. What remains is to show that the approximation ratio of the solution f_{apx} is bounded by $1+\epsilon$.

Let D_d be a chunk decomposition of G. A layer L is called a "boundary layer" for D_d if L is either one of the first $2h$ layers or one of the last $2h$ layers in a chunk in D_d. Note that a boundary layer is either an empty layer (if it is one of the first $2h$ layers in the 0th chunk or one of the last $2h$ layers in the last chunk in the decomposition D_d), or is shared by two consecutive chunks in D_d. By the construction of the chunk decompositions, every layer in $\pi(G)$ is a boundary layer for exactly one chunk decomposition. Therefore, the layers in $\pi(G)$ can be partitioned into disjoint layer sets S_i, $0 \leq i \leq q/(2h)-2$, where S_i consists of all boundary layers in the chunk decomposition D_i.

Now suppose that f_{opt} is a minimum weight assignment of weight $|f_{opt}|$ to the input variables in G_w that satisfies the circuit G_w. Let V_{opt} be the set of input variables which are assigned value 1 by f_{opt} (thus, $|f_{opt}| = |V_{opt}|$). Since the layer sets S_i, $0 \leq i \leq q/(2h)-2$, are disjoint, one of the layer sets contains at most

$|f_{opt}|/(q/(2h)-1) \le \epsilon \cdot |f_{opt}|$ input variables in V_{opt}. Let this set be S_d, and let V_{opt}^d be the set of input variables in both V_{opt} and S_d, $|V_{opt}^d| \le \epsilon \cdot |f_{opt}|$. We consider the chunk decomposition D_d. Let f_d be the assignment to the input variables in G_w constructed by our algorithm based on the chunk decomposition D_d.

Let U_0, U_1, ..., U_p be the chunks in D_d. Let f_{d,U_i} be the input assignment we construct for the chunk U_i, and let f_{opt,U_i} be the input assignment in U_i induced from f_{opt}. Note that the assignment f_{opt,U_i} also satisfies all depth-$(h-1)$ gates in the middle $q-2h$ layers in U_i, and by our construction, f_{d,U_i} is a minimum weight assignment that satisfies all depth-$(h-1)$ gates in the middle $q-2h$ layers in U_i. Thus, if we let $|f_{opt,U_i}|$ and $|f_{d,U_i}|$ be the weights of these two assignments, we have $|f_{opt,U_i}| \ge |f_{d,U_i}|$. Therefore,

$$\sum_{i=0}^{p} |f_{opt,U_i}| \ge \sum_{i=0}^{p} |f_{d,U_i}|$$

Since for each input variable v, we have $f_d(v) = f_{d,U_i}(v)$ if v is in the middle $q-2h$ layers of the chunk U_i, and $f_d(v) = f_{d,U_i}(v) \vee f_{d,U_{i+1}}(v)$ if v is in a boundary layer shared by two chunks U_i and U_{i+1}, we have $\sum_{i=0}^{p} |f_{d,U_i}| \ge |f_d|$. Moreover, in the summation $\sum_{i=0}^{p} |f_{opt,U_i}|$, each input variable in the set V_{opt}^d counts exactly twice and each input variable in $V_{opt} - V_{opt}^d$ counts exactly once, thus

$$\sum_{i=0}^{p} |f_{opt,U_i}| = |f_{opt}| + |V_{opt}^d| \le |f_{opt}|(1+\epsilon)$$

Finally, since the assignment f_{apx} constructed by our algorithm is the assignment f_d with minimum weight over all d, we derive immediately:

$$|f_{apx}| \le |f_d| \le \sum_{i=0}^{p} |f_{d,U_i}| \le \sum_{i=0}^{p} |f_{opt,U_i}| \le |f_{opt}|(1+\epsilon)$$

and conclude that the approximation ratio of our algorithm is bounded by $1+\epsilon$. □

We briefly describe how Lemma II.2 and Theorem II.4 are modified to apply to the classes PLANAR MAX-$W[h]$ and PLANAR $W[h]$-SAT.

Given an instance G_w of a PLANAR MAX-$W[h]$ problem and a real number $\epsilon > 0$, where G_w is a planar Π_h^--circuit with the output gate w, we let $q = h(\lceil 1/\epsilon_0 \rceil + 1)$, where $\epsilon_0 = \epsilon/(1+\epsilon)$, and construct a planar embedding $\pi(G)$ of the graph $G = G_w - w$. Now each chunk decomposition D_d partitions the graph G into *disjoint* chunks, each consists of q consecutive layers in $\pi(G)$. The first h layers and the last h layers in a chunk will be called the *boundary layers* of the chunk. Assign value 1 to input gates that are in boundary layers of the chunks. Since all input gates are labeled by negations of input variables, this assignment is equivalent to assigning value 0 to the corresponding input variables. According to this assignment, if a gate g_1 has an input from a gate g_2 such that g_1 and g_2 belong to two different chunks, then the gate g_2 must have value 1 since all input gates that can affect the gate g_2 are in boundary layers and hence have been assigned value 1.

With this initial assignment, now we work on each chunk U in D_d. Note that it is always possible to assign the remaining input gates in the chunk U to satisfy all depth-$(h-1)$ gates in U (e.g., assigning all remaining input gates in U value 1, or equivalently, assigning all remaining input variables in U value 0). Since the chunk U is given as its q-outerplanar embedding induced from $\pi(G)$, using the techniques similar to that of Lemma II.2, we can construct a maximum weight assignment to the remaining input variables in U that satisfies all depth-$(h-1)$ gates in U. As shown in Lemma II.2, such an assignment can be constructed in time $O(81^q n_U)$, where n_U is the number of vertices in U. Doing this for all chunks in the chunk decomposition D_d gives an assignment f_d to the input variables that satisfies all depth-$(h-1)$ gates in G_w, thus satisfying the circuit G_w. Now we apply this process to each possible chunk decomposition D_d, each gives an assignment f_d satisfying the circuit G_w. We pick the one, denoted by f_{apx}, with the largest weight among all f_d's and output it as the approximation solution to the problem. The assignment f_{apx} can be constructed in time $O(81^q n^2) = O(81^{O(1/\epsilon)} n^2)$.

Similar to the analysis given in Theorem II.4, if we fix an optimal assignment f_{opt} to the circuit G_w, then there is a chunk decomposition D_d in which the number m_d of variables that are in the boundary layers of D_d and are assigned value 1 by the assignment f_{opt} is bounded by $\epsilon_0|f_{opt}|$. Moreover, the weight of the assignment f_d constructed based on the chunk decomposition D_d is at least $|f_{opt}| - m_d$. Since the weight of the assignment f_{apx} is the largest among all f_d's, we conclude that the assignment f_{apx} has weight at least $|f_{opt}| - m_d$. In consequence, the ratio $|f_{apx}|/|f_{opt}|$ is at least $1-\epsilon_0$. Replacing ϵ_0 by $\epsilon/(1+\epsilon)$ gives the approximation ratio $|f_{opt}|/|f_{apx}| \leq 1 + \epsilon$. This completes the proof that every problem in PLANAR MAX-$W[h]$ is in the class EPTAS.

The EPTAS algorithm for a problem in PLANAR $W[h]$-SAT is similar. Again we use chunk decompositions of disjoint chunks, but do not apply any initial assignments. For each chunk U, we construct an assignment to the input variables in U to satisfy the largest number of depth-$(h-1)$ gates that are in the middle $q-2h$ layers in U (note that no input gates outside chunk U can affect these gates). We leave the verification of the details to the interested readers.

Theorem II.5 *For every $h \geq 1$,* PLANAR MAX-$W[h]$ *and* PLANAR $W[h]$-SAT *are subclasses of the class EPTAS.*

Cesati and Trevisan [17] proved that if an optimization problem is in the class EPTAS then its parameterized version is fixed-parameter tractable. Combining this with Theorem II.4 and Theorem II.5, we get the following:

Corollary II.6 *For every positive integer h, the classes* PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, *and* PLANAR $W[h]$-SAT *are subclasses of FPT.*

D. Remarks

In this chapter, under a very general constraint of scalability, we presented a precise characterization of the approximation class FPTAS in term of its parameterized complexity, the efficient fixed-parameter tractability. This new characterization has a number of advantages over the previous characterizations of the approximation class FPTAS.

Not enough attention has been paid to the computational complexity of general PTAS algorithms for NP optimization problems. Many developments of PTAS algorithms simply sought a polynomial bound on the running time of the algorithms with the hope that once a polynomial time approximation algorithm is derived, it will sooner or later be improved to become practically efficient. The recent progress in the study of parameterized computation has shown that this understanding is not always correct: unless an unlikely collapse in parameterized complexity theory occurs, there are PTAS problems for which any PTAS algorithm must have the time complexity in which $1/\epsilon$ is in the exponent of the input size n [14, 46]. In particular, our very recent research has shown that under a similar conjecture, there are PTAS problems in computational biology for which there is a constant $c > 0$ such that any PTAS algorithms for these problems must have time complexity of order $\Omega(n^{c/\epsilon})$ [22]. For this kind of PTAS problems, practically efficient polynomial-time approximation schemes are unlikely even for moderate values of the approximation error bound ϵ.

The introduction of the concept of EPTAS attempts to refine the class PTAS and characterize the PTAS problems that admit practically efficient polynomial-time approximation schemes. Since the initialization of this line of research [12, 14, 17], we make the first attempt to a systematic investigation of the structural properties of this new but important approximation class. Based on the fixed-parameter intractable hierarchy, the W-hierarchy, and by enforcing a planarity constraint, we presented the syntactic classes, PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT, and showed that all problems in these classes belong to the approximation class EPTAS. These syntactic classes seem to form the core for a very significant class of EPTAS problems.

We point out that our syntactic classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT are significantly different from the classes TMIN, TMAX, and MPSAT proposed by Khanna and Motwani [57] in the following sense. First, Corollary II.6 proves that all optimization problems in our syntactic classes are fixed-parameter tractable, while Cai et. al [14] recently proved that there are $W[1]$-hard problems in the syntactic classes introduced in [57]. This shows that these problems cannot be contained in our syntactic classes unless an unlikely collapse in parameterized complexity theory occurs. On the other hand, our classes are not subclasses of that of Khanna and Motwani's: the classes TMIN, TMAX, and MPSAT are defined based on circuits of depth 3, while ours are defined based on circuits of any constant depth. According to the well-known research on constant depth circuits [48], the classes PLANAR $W[h]$-SAT, PLANAR MIN-$W[h]$, and PLANAR MAX-$W[h]$ for $h > 3$ *cannot* be expressed by the syntactic classes TMIN, TMAX, and MPSAT.

CHAPTER III

LOWER BOUNDS OF PARAMETERIZED COMPUTATION*

In this chapter, based on the study of the structural properties of parameterized complexity, we develop new techniques for deriving strong computational lower bounds for a class of well-known NP-hard problems and some Non NP-hard problems. The NP-hard problems include WEIGHTED SATISFIABILITY, DOMINATING SET, HITTING SET, SET COVER, CLIQUE, and INDEPENDENT SET. And the Non NP-hard problems are the problems in the class LOGNP, such as TOURNAMENT DOMINATING SET and V-C DIMENSION.

A. Parameterized NP-hard Problems

The result reported here in this section is joint work with J. Chen, I. Kanj, and G. Xia [23].

1. Introduction

In parameterized computation, fixed-parameter tractable algorithms, whose running time takes the form $f(k)n^{O(1)}$ for a function f, have been used to solve a variety of difficult computational problems in practice. The concept of $W[1]$-hardness has been introduced to address fixed-parameter intractability, and a large number of $W[1]$-hard parameterized problems have been identified [37]. Now it has become commonly accepted that no $W[1]$-hard problem can be solved in time $f(k)n^{O(1)}$ for any function f (i.e., $W[1] \neq FPT$).

---

*Part of the data reported in this chapter is reprinted with permission from "Linear FPT reductions and computational lower bounds" by J. Chen, X. Huang, I. Kanj, and G. Xia, 2004, *Proceedings of the 36th ACM Symposium on Theory of Computing* (SOTC 2004), pp. 212-221, Copyright 2004 by ACM.

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]

Take the $W[1]$-hard parameterized problem CLIQUE as an example. We know that a trivial enumeration can easily test in time $O(n^k)$ if a given graph of n vertices has a clique of size k. Is it possible for it to have algorithms of uniform running time $n^{o(k)}$? Can the problem be solvable in time $n^{o(k)}$ for an extreme range of the parameter values such as $k = \log\log n$ or $k = n^{4/5}$? Moreover, is it possible that the problem be solvable in time $f(k)n^{o(k)}$ for a function f?

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 greatly improve the results in [21]. 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

[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 $O(n^{\log\log k})$.

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 [65] 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 $m^{o(k)}$.

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 [1, 2, 26, 29, 63]. Continuously improved algorithms for these problems have been developed based on improved local structure manipulations (for example, see [78, 54, 71, 9] on the progress for the INDEPENDENT SET problem). It has even been proposed to automate the manipulation of local structures [64, 72] 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.

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 $f(n) = 2^{o(n)}$.

2. 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.

Recall the definitions given in the Chapter I: A circuit C is a Π_t-*circuit* if its output gate is an AND gate and it has depth t. The SATISFIABILITY problem on Π_t-circuits, abbreviated SAT$[t]$, is to determine if a given Π_t-circuit C has a satisfying assignment. WCS$^*[t]$ is the problem WCS$^+[t]$ if t is even and the problem WCS$^-[t]$ if t is odd, where WCS$^+[t]$ and WCS$^-[t]$ are the WEIGHTED MONOTONE SATISFIABILITY problem and the WEIGHTED ANTIMONOTONE SATISFIABILITY problem respectively on Π_t-circuits.

Lemma III.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 $1 \leq i \leq k-1$, and $|B_k| \leq r$. For an integer j, $0 \leq j \leq 2^{|B_i|} - 1$, 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 $z_{i,0}$, $z_{i,1}$, …, $z_{i,s-1}$. 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 $0 \leq j \leq 2^{|B_i|} - 1$ 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 $0 \leq j \leq 2^{|B_i|} - 1$ 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 $\mathrm{bin}_i(j_i) = \tau_i$. 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 $\tau_i = \mathrm{bin}_i(j_i)$. 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 $\mathrm{bin}_i(j)$ 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 $\mathrm{bin}_i(j)$ 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 $\text{bin}_i(j_i)$ is 0, or $\overline{x}_q$ is an input to g_1 in C_1 and the h-th bit of $\text{bin}_i(j_i)$ is 1. However, by our construction of the index j_i from the assignment τ, if the h-th bit of $\text{bin}_i(j_i)$ is 0 then τ assigns $x_q = 0$, and if the h-th bit of $\text{bin}_i(j_i)$ 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 $\tau_i = \text{bin}_i(j_i)$ 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 III.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 antimonotone 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 III.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 $0 \leq h_i \leq |B_i|$ 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, $1 \leq i \leq k_2$, 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. $\square$

3. Lower Bounds on Weighted Satisfiability Problems

From Lemma III.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 III.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 III.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 III.1, we can determine if C_1 is a yes-instance of SAT$[t]$ by calling the algorithm M_{WCS} to determine if (C_2, k) is a yes-instance of WCS$^*[t]$. The running time of the algorithm M_{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_{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_{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 $k \le f^{-1}(n_1) \le \log n_1$),

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

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_{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_{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. $\square$

In fact, Theorem III.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 III.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 III.3. $\square$

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

Theorem III.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 III.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

$$\mu\left(\frac{n_1 2^{z^* \log n_1}}{z^* \log n_1}\right) \leq \frac{n_1}{z^* \log n_1} \qquad \text{and} \qquad \mu\left(\frac{n_1 2^{z^* \log n_1 + 1}}{z^* \log n_1 + 1}\right) \geq \frac{n_1}{z^* \log n_1 + 1} \quad (3.1)$$

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

$$\begin{aligned} 2\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) &\ge 2\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1 + 1}\right) \ge \mu\left(\frac{n_1 2^{z^*\log n_1 + 1}}{z^*\log n_1 + 1}\right) \\ &\ge \frac{n_1}{z^*\log n_1 + 1} \ge \frac{n_1}{2z^*\log n_1} \end{aligned} \tag{3.2}$$

where the second inequality uses the fact $2\mu(n) \ge \mu(2n)$. From (3.1) and (3.2), we get

$$4\mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) \ge \frac{n_1}{z^*\log n_1} \ge \mu\left(\frac{n_1 2^{z^*\log n_1}}{z^*\log n_1}\right) \tag{3.3}$$

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 (3.3), 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 (3.1). As we explained above, the value z^* can be computed in time polynomial in n_1;
>
> (B) Call the algorithm A_1 in Lemma III.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 $m_2 \le 2m_1 + 2^{2r+1}k$. By the above discussion, we have $\mu(n_2)/8 \le k \le 16\mu(n_2)$;
>
> (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 III.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),

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

Now it is easy to verify that $n_2^{k/\lambda(k)} = 2^{o(n_1)}$ (observe that $k = n_1/r \ge \mu(n_2)/8$ hence $\lambda(n_1/r)$ is unbounded, and that $r = z^* \log n_1 = \Omega(\log n_1)$). Also, since $m_2 \le 2m_1 + 2(n_2)^2$, $m_2 = 2^{o(n_1)} m_1^{O(1)}$, 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. $\square$

Now we derive similar results for the weighted satisfiability problem WCNF 2-SAT$^-$, based on Lemma III.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 III.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 III.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 III.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 $2^{o(n)}$. $\square$

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

Corollary III.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 III.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 III.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 III.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 III.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 $2^{o(n_1)}$. $\square$

4. Satisfiability Problems and the W-hierarchy

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

Theorem III.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 [15]. 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 $\mathrm{bin}_r(j)$ 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 $\mathrm{bin}_r(j)$ 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 $\mathrm{bin}_r(j)$ 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 $\mathrm{bin}_r(j)$ 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 $x_{j_1}, \ldots, x_{j_k}$, 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' 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 $\overline{x}_{j_1}, \ldots, \overline{x}_{j_k}$ are inputs to g_1 in C, no gates $g_{1,j_1}, \ldots, g_{k,j_k}$ 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 $\text{bin}_r(j_i) = \tau'_i$. 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 $1 \leq i \leq k$ 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 $t \geq 3$, for a given Π_{t-1}-circuit C of n input variables and size m, and for a given $k \leq n$, 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 $m + kn + 3k^2 \log^2 n \leq 3m^3$, where the term kn is the number of the gates $g_{i,j}$, $1 \leq i \leq k$, $0 \leq j \leq n-1$ 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 $W[t-1] = FPT$. $\square$

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

Theorem III.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 III.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 [65] 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 [53]. It is commonly believed that it is unlikely that all problems in SNP are solvable in subexponential time[3]. Impagliazzo and Paturi [53] 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.

[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* [34].

Lemma III.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 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 [53], this in consequence implies that all problems in SNP are solvable in subexponential time. $\square$

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

Theorem III.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* $k \le \mu(n)$ *for any nondecreasing and unbounded function* μ*.*

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

5. 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 [26]). In the following, we present a simple algorithm for reducing the search space size for the DOMINATING SET problem (given a graph G and an integer k, decide whether there is a dominating set of k vertices, i.e., a subset D of k vertices such that every vertex not in D is adjacent to at least one vertex in D).

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 III.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 III.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.

The parameterized problems discussed here 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). 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 III.10 and III.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 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'.

From the definition of fpt$_l$-reduction, the transitivity of the fpt$_l$-reduction can be easily deduced:

Lemma III.16 *Let Q_1, Q_2, and Q_3 be three parameterized problems. If Q_1 is fpt$_l$-reducible to Q_2, and Q_2 is fpt$_l$-reducible to Q_3, then Q_1 is fpt$_l$-reducible to Q_3.*

PROOF. If Q_1 is fpt$_l$-reducible to Q_2, then there exist a function f_1 and an algorithm A_1 of running time $f_1(k_1)n_1^{o(k_1)}m_1^{O(1)}$, such that on each (k_1, n_1, m_1)-instance x_1 of Q_1, the algorithm A_1 produces a (k_2, n_2, m_2)-instance x_2 of Q_2, where $n_2 = n_1^{O(1)}$, $m_2 = m_1^{O(1)}$, and $k_2 \leq c_1 k_1$, where c_1 is a constant.

If Q_2 is fpt$_l$-reducible to Q_3, then there exist a function f_2 and an algorithm A_2 of running time $f_2(k_2)n_2^{o(k_2)}m_2^{O(1)}$, such that on each (k_2, n_2, m_2)-instance x_2 of Q_2, the algorithm A_2 produces a (k_3, n_3, m_3)-instance x_3 of Q_3, where $k_3 = O(k_2)$, $n_3 = n_2^{O(1)}$, $m_3 = m_2^{O(1)}$.

Now we have an algorithm A that reduces Q_1 to Q_3, as follows. For a given (k_1, n_1, m_1)-instance x_1 of Q_1, A first calls the algorithm A_1 on x_1 to constructs a (k_2, n_2, m_2)-instance x_2 of Q_2, where $k_2 \leq c_1 k_1$, $n_2 = n_1^{O(1)}$, and $m_2 = m_1^{O(1)}$. Then A calls the algorithm A_2 on x_2 to construct a (k_3, n_3, m_3)-instance x_3 of Q_3. It is obvious that x_3 is a yes-instance of Q_3 if and only if x_1 is a yes-instance of Q_1. Moreover, from $k_2 \leq c_1 k_1$ and $k_3 = O(k_2)$, we have $k_3 = O(k_1)$, and from $n_2 = n_1^{O(1)}$, $m_2 = m_1^{O(1)}$, $n_3 = n_2^{O(1)}$, $m_3 = m_2^{O(1)}$, we get $n_3 = n_1^{O(1)}$, $m_3 = m_1^{O(1)}$. Finally, since the call to algorithm A_1 on x_1 takes time $f_1(k_1)n_1^{o(k_1)}m_1^{O(1)}$, the call to algorithm A_2 on x_2 takes time $f_2(k_2)n_2^{o(k_2)}m_2^{O(1)}$, and $k_2 \leq c_1 k_1$, $n_2 = n_1^{O(1)}$, and $m_2 = m_1^{O(1)}$, we conclude that the running time of the algorithm A is bounded by $f(k_1)n_1^{o(k_1)}m_1^{O(1)}$, where $f(k_1) = f_1(k_1) + f_2(c_1 k_1)$. By the definition, A is an fpt$_l$-reduction from Q_1 to

Q_3, i.e., Q_1 is fpt_l-reducible to Q_3. □

Definition 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 III.10 and Theorem III.13, we immediately derive:

Theorem III.17 *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 III.18 *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 III.19 *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 III.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. □

We remark that the reduction from WCS$^+[2]$ to DOMINATING SET presented in the proof of Theorem III.19 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 [37].

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 III.20 *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 III.19 and Theorem III.20 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 [1, 2, 26, 29, 63]. 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 [64, 72] in order to further improve the computational time.

Theorem III.19 and Theorem III.20, 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 III.19 and the lower bound $f(k)m^{\Omega(k)}$ for the problems in Theorem III.20, 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.

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 III.19 and Theorem III.20 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 III.21 *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 III.19 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 III.19, each fpt$_l$-reduction from wcs$^+[2]$ to a problem in Theorem III.19 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 III.11. $\square$

Note that the conditions on the function μ in Theorem III.21 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 III.20, by similar proofs based on Theorem III.14.

Theorem III.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 III.20 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 III.19 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].

Theorem III.23 *All problems in Theorem III.19 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 [37], 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 left to the interested readers. The theorem now follows from the transitivity of the fpt$_l$-reduction. □

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

B. On Some Parameterized Non NP-hard Problems

The work of this section is motivated by our study on the computational lower bounds for the parameterized NP-hard problems via the definition of linear fpt-reduction. We study the problems in the class LOGNP introduced by Papadimitriou and Yannakakis [66]. Since these problems can be solved deterministically in time $O(n^{\log n})$, they are unlikely to be NP-hard. We prove lower bound results for the problems in the class LOGNP.

1. Further Remarks on $W_l[1]$-hardness

We have given the definition of fpt$_l$-reduction and based on it defined W$_l$[1]-hardness. We proved that no W$_l$[1]-hard problem can be solved in time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.

Let Q be a parameterized problem and let r be any nondecreasing and unbounded function, we define a subset r-Q of Q:

$$r\text{-}Q = \{(x,k) \mid (x,k) \in Q \ \text{ and } \ k \leq r(|x|)\}$$

We have the following theorem.

Theorem III.24 *For a W_l[1]-hard problem Q solvable in time $O(c^n)$ for a constant c and for any nondecreasing and unbounded function r, the problem r-Q has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. Suppose the problem r-Q is solvable by an algorithm A of running time $f(k)n^{o(k)}$ for a recursive function f.

Define the function r^- as $r^-(p) = max\{r(q) \leq p\}$. Since r is non-decreasing and unbounded, r^- is also a non-decreasing and unbounded function. Define $f'(k) = c^{r^-(k)}$. Consider the following algorithm A' solving Q:

For a given instance (x, k) of the problem Q, if $k \geq r(n)$, then solve the problem in time $O(c^n)$; and if $k < r(n)$, call the algorithm A to solve the problem.

We claim that the algorithm A' solves the problem Q in its general case in time $F(k)n^{o(k)}$, where F is a function to be decided. In fact, in case $k \geq r(n)$, we have $r^-(k) \geq n$, therefore $f'(k) \geq c^n$. Thus, the running time of the algorithm A' is bounded by $O(c^n) = O(f'(k)) = f'(k)n^{o(k)}$. On the other hand, in case $k < r(n)$, by the hypothesis of the theorem, the algorithm A runs in time $f(k)n^{o(k)} \leq max(f(k), f'(k))n^{o(k)} \leq F(k)n^{o(k)}$, where $F(k) = max(f(k), f'(k))$. Thus, the running time of the algorithm A' is always bounded by $F(k)n^{o(k)}$.

By Theorem III.18, the existence of the algorithm A' of time $F(k)n^{o(k)}$ for the $W_l[1]$-hard problem Q would imply all SNP problems are solvable in subexponential time. □

We prove the following theorem:

Theorem III.25 *Suppose that a problem Q_1 has no algorithm of time $f(k)n^{o(k)}$ for any function f, and that Q_1 is fpt$_l$-reducible to Q_2. Then the problem Q_2 has no algorithm of time $f'(k)n^{o(k)}$ for any function f'.*

PROOF. Assume the problem Q_2 has an algorithm A' of time $f'(k)n^{o(k)}$ for a recursive function f'. We have the following algorithm A for the problem Q_1:

Given an instance (x_1, k_1) of the problem Q_1, by the fpt$_l$-reduction, we reduce it in time $f_l(k_1)n_1^{o(k_1)}$ to an instance (x_2, k_2) of the problem Q_2, where f_l is a recursive function, $k_2 \leq c_1 k_1$ with a constant c_1, $n_2 = n_1^{O(1)}$. Call the algorithm A' on the instance (x_2, k_2) and return "yes" if A' returns "yes"; Otherwise return "no".

The reduction takes time $f_l(k_1)n_1^{o(k_1)}$. And the call to the algorithm A' takes time $f'(k_2)n_2^{o(k_2)} \leq f'(c_1k_1)n_1^{o(k_1)}$. Therefore we have the algorithm A for the problem Q_1 of time bounded by $f(k)n^{o(k)}$, where $f(k) = f_l(k) + f'(c_1k)$. This causes a contradiction. Our assumption is not correct. The theorem is proved. □

2. Parameterized LOGNP Problems

We have demonstrated that for NP-hard optimization problems we can derive strong computational lower bounds. In this section, we give a uniform method to prove lower bound results for some Non NP-hard problems in the class LOGNP.

The problems in the class LOGNP [66] are decision problems. First we give the definitions of the standard parameterized LOGNP problems and then derive lower bounds for these parameterized problems.

> LOG ADJUSTMENT-PARA: given a Boolean expression F in conjunctive normal form with n variables, and a truth assignment T, and a parameter k, where $k \leq \log n$, is there a satisfying truth assignment whose Hamming distance from T is k?

A chordless path of a graph G is a simple path $v_1, v_2, ..., v_n$, such that on this path any two vertices v_i and v_j with $|i - j| > 1$ are not adjacent.

> LOG CHORDLESS PATH-PARA: given a graph $G = (V, E)$, where $|V| = n$, and a parameter k, where $k \leq \log n$, is there a chordless path of length k?

> LOG CLIQUE-PARA: given a graph $G = (V, E)$, where $|V| = n$, and a parameter k, where $k \leq \log n$, is there a clique of size k?

> LOG DOMINATING SET-PARA: given a graph $G = (V, E)$, where $|V| = n$, and a parameter k, where $k \leq \log n$, is there a dominating set of size k?

LOG HYPERGRAPH COVER-PARA: given a hypergraph $H = (V, E)$, where $|V| = n$, and a parameter k, where $k \leq \log n$, is there a vertex cover of size k for H?

RICH HYPERGRAPH COVER-PARA: given a hypergraph $H = (V, E)$, where $|V| = n$ and all edges of size at least $n/2$, and a parameter k, where $k \leq \log n$, is there a vertex cover of size k for H?

A tournament graph is a directed graph $G = (V, E)$, where for any two vertices $u, v \in V$, $u \neq v$, exactly one of the directed edge (u, v) or (v, u) is in E.

TOURNAMENT DOMINATING SET-PARA: given a tournament graph G, and a parameter k, is there a dominating set of size k for G?

V-C DIMENSION-PARA: given a family C of subsets of a universe U, and a parameter k, is there a subset S of U such that $|S| = k$ and for each subset T of S, there is a set $C_T \in C$ satisfying $S \cap C_T = T$?

All these problems can be solved in time $O(n^{\log n})$. They are unlikely to be NP-hard since otherwise, all NP problems could be solved in time $O(n^{O(\log n)})$. But none of them are known to be solvable in polynomial time.

We first prove lower bound results for the LOG CLIQUE-PARA and LOG DOMINATING SET-PARA.

Theorem III.26 LOG CLIQUE-PARA *and* LOG DOMINATING SET-PARA *cannot be solved in time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. By our notation of r-Q, LOG CLIQUE-PARA is $\log n$-CLIQUE, and LOG

DOMINATING SET-PARA is $\log n$-DOMINATING SET. From Theorem III.20 and Theorem III.23, CLIQUE and DOMINATING SET are $\mathrm{W}_l[1]$-hard. By Theorem III.24, this theorem is true. □

We now prove lower bound results for other problems in the class LOGNP.

Theorem III.27 LOG HYPERGRAPH COVER-PARA *cannot be solved in time* $f(k)n^{o(k)}$ *for any function* f*, unless all SNP problems are solvable in subexponential time.*

PROOF. We give an fpt_l-reduction from LOG DOMINATING SET-PARA to LOG HYPERGRAPH COVER-PARA. By Theorem III.25 and Theorem III.26, the theorem follows.

The fpt_l-reduction is adapted from the polynomial time reduction in [66]. Given an instance (G, k) of LOG DOMINATING SET-PARA, where $G = (V, E)$ and $k \le \log n$, we construct a hypergraph H. H has the same vertex set V as G. For each vertex v of G, we build a hyperedge e_v, which contains the vertex v and all its neighbors in G.

Suppose the graph G has a dominating set S. For each vertex v of G, either $v \in S$ or v has a neighbor $u \in S$. From the construction of the hypergraph H, we can see that for each hyperedge e_v, it is covered either by v or v's neighbor u in G. So, S is a cover of the hypergraph H. On the other hand, suppose S is a cover of the hypergraph H, then for each hyperedge e_v, $v \in S$ or $u \in S$, where $(u, v) \in E$. Since for each vertex $v \in V$ we have built a hyperedge e_v, then we know for each vertex $v \in V$, either $v \in S$ or one of its neighbor u in S. The vertex set S is a dominating set for G. Therefore, the graph G has a dominating set of size k if and only if there is a cover of size k for the hypergraph H. The reduction is an fpt_l-reduction. □

Theorem III.28 LOG ADJUSTMENT-PARA *cannot be solved in time* $f(k)n^{o(k)}$ *for any function* f*, unless all SNP problems are solvable in subexponential time.*

PROOF. We give an fpt_l-reduction from LOG HYPERGRAPH COVER-PARA to LOG ADJUSTMENT-PARA. By Theorem III.25 and Theorem III.27, the theorem follows.

The fpt_l-reduction is adapted from the polynomial time reduction in [66]. Suppose we are given an instance (H, k) of the LOG HYPERGRAPH COVER-PARA, where $H = (V, E)$ is a hypergraph with $|V| = n$, and k is a parameter with $k \leq \log n$. We will construct an instance (F, T, k) of LOG ADJUSTMENT-PARA. We build F as a conjunctive normal form with n positive input variables $\{v_1, v_2, ..., v_n\}$. The n positive input variables represent the n vertices of H. Each clause of F, which corresponds to an edge e of the hypergraph H, is a disjunction of all the variables that represent the vertices of the edge e. We assign all variables FALSE as the default truth assignment T.

Suppose H has a cover C of size k. For each edge $e \in E$, at least one of its vertices, say v, is in C. Then in F, for the clause that corresponds to the edge e, we assign TRUE to the variable that corresponds to the vertex v. So, F is satisfied by a truth assignment T' with all variables corresponding to the vertices in C being assigned TRUE and the other variables being assigned FALSE. The Hamming distance between T' and T is k. On the other hand, suppose there is a satisfying truth assignment T' whose Hamming distance from T is k. We can get a cover for H, which contains the vertices that correspond to all the variables with TRUE values in T'. Therefore, there is a cover of size k for H if and only if there is a satisfying truth assignment whose Hamming distance from T is k. The reduction is an fpt_l-reduction. □

Theorem III.29 RICH HYPERGRAPH COVER-PARA *cannot be solved in time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. We give an fpt$_l$-reduction from LOG HYPERGRAPH COVER-PARA to RICH HYPERGRAPH COVER-PARA. By Theorem III.25 and Theorem III.27, the theorem follows.

The fpt$_l$-reduction is essentially the same as the polynomial time reduction in [66]. From an instance of LOG HYPERGRAPH COVER-PARA $\langle H = (V, E), k\rangle$, where $|V| = n$. we construct an instance of RICH HYPERGRAPH COVER-PARA $\langle H' = (V', E'), k\rangle$. The RICH HYPERGRAPH COVER-PARA problem requires all the edges contain at least half of the vertices of the graph. The edges of H may not satisfy this requirement. As in [66], we will construct H' by taking copies of the edges of H and adding new vertices to enlarge them.

Let $l = 3 \log n$ and $r = (2^l - 1)^2$. Every integer i, $1 \leq i \leq r$, could be interpreted as a binary vector of the form a_1a_2, where a_1 and a_2 are nonzero vectors of length l. V' contains all the vertices in V and r new vertices $u_1, ..., u_r$. For every edge $e \in E$, we construct r edges $e_1, ..., e_r$. Each of the r edges contains the same set of original vertices as e and also include $3/4$ of the r new vertices as follows: suppose i corresponds to the vector a_1a_2 and j corresponds to the vector b_1b_2, for each new vertex u_i, $1 \leq i \leq r$, it belongs to the edge e_j if and only if the inner product $a_1 \cdot b_1 = 1$ or $a_2 \cdot b_2 = 1$, where the arithmetic is in GF(2), i.e., $0+0 = 1+1 = 0$, $0+1 = 1+0 = 1$, $0 \times 0 = 0 \times 1 = 1 \times 0 = 0$, $1 \times 1 = 1$. This finishes the construction of H'.

Now we show that H has a cover of size k if and only if H' has a cover of size k. Suppose H has a cover of size k. By the construction of H', each edge of H' contains the same vertex set as one edge of H. So H' has the same cover as H.

On the other hand, if H' has a cover C' of size k, $k < l$. We can prove that the "old" vertices in C' form a cover C of H, i.e., $C = \{v : v \in C' \text{ and } v \in V\}$, as follows: Suppose there is an edge $e \in E$ not covered by any old vertex. consider the r edges $e_1, ..., e_r$ in H' that correspond to the edge e. There is at least one edge e_j of the r

edges, such that for any "new" vertex $u_i \in C'$, $a_1 \cdot b_1 = 0$ and $a_2 \cdot b_2 = 0$, where i corresponds to the vector $a_1 a_2$ and j corresponds to the vector $b_1 b_2$ (since $|C'| < l$, there are less than l values of a_1 and a_2). So, the edge e_j is not covered by any old or new vertex in C'. If $|C| < k$, we can randomly add some vertices into the cover C to make its size equal to k. Therefore, H has a cover of size k if and only if H' has a cover of size k. The reduction is an fpt$_l$-reduction. □

Theorem III.30 LOG CHORDLESS PATH-PARA *cannot be solved in time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. We give an fpt$_l$-reduction from LOG CLIQUE-PARA to LOG CHORDLESS PATH-PARA. By Theorem III.25 and Theorem III.26, the theorem follows.

The fpt$_l$-reduction is adapted from the polynomial time reduction in [66]. Given an instance (G, k) of LOG CLIQUE-PARA, where the graph $G = (V, E)$ with n vertices, we construct a graph G' as follows: First, G' has k disjoint copies of V; the jth copy V_j has vertices c_{ij}, $i = 1, ..., n$. Two vertices c_{ij} and $c_{i'j'}$ are connected in G' if and only if $i = i'$ or $j = j'$ or $(i, i') \notin E$. Finally, for all $j < k$ we have a path of length two (p_{j1}, p_{j2}, p_{j3}) and edges from all vertices of V_j to p_{j1} and from p_{j3} to all vertices of V_{j+1}.

We show that G has a clique of size k if and only if there is a chordless path of length k', $k' = 4(k-1)$. If G has a clique of size k, then by taking a copy of its vertices, one from each copy of V, and connecting them in order via the paths of length four, we form a chordless path of length $4(k-1)$ vertices. On the other hand, suppose that G' has a chordless path P of length k'. Since every copy V_j of V induces a clique, P cannot contain more than two vertices from the same copy, and if it does contain two vertices then it cannot contain the vertices p_{j1}, p_{j2}, p_{j3} of the following and the preceding length-two path. It follows from this observation that for P to have length $k' = 4(k-1)$, it must contain all the length-two paths and exactly one vertex from each copy of V. Then the i indices of the vertices of P in the copies of V must form a clique of the graph G, and there are k of them. The reduction is an fpt$_l$-reduction. □

Theorem III.31 V-C DIMENSION-PARA *cannot be solved in time* $f(k)n^{o(k)}$ *for any function* f*, unless all SNP problems are solvable in subexponential time.*

PROOF. We give an fpt$_l$-reduction from CLIQUE to V-C DIMENSION-PARA. The fpt-reduction from CLIQUE to V-C DIMENSION-PARA in [35] for proving V-C DIMENSION-PARA is W[1]-complete is essentially an fpt$_l$-reduction.

Given a graph $G = (V, E)$, $V = \{1, ..., n\}$, and an integer $k > 0$, we construct a family of sets F over a base set X, so that F has V-C dimension k if and only if G has a k-clique.

The base set X is:

$$X = \{(u, i) : u \in V, 1 \leq i \leq k\}.$$

The size of the base set X is kn.

The family F consists of four subfamilies, $F = F_0 \cup F_1 \cup F_2 \cup F_3$, where

$$F_0 = \{\phi\},$$

$$F_1 = \{\{(u, i)\} : u \in V, 1 \leq i \leq k\},$$

$$F_2 = \{\{(u, i), (v, j)\} : [u, v] \in E, 1 \leq i, j \leq k\},$$

$$F_3 = \{\{(u, i) : u \in V, i \in S\} : S \subseteq \{1, 2, \ldots, k\}, |S| \geq 3\}.$$

The family F_0 has one set, the family F_1 has nk sets, the family F_2 has $mk^2 = O(n^2k^2)$

sets, and the family F_3 has $\sum_{i=3}^{k}\binom{k}{i} = O(2^k)$ sets. Therefore, the cardinality of the family F is $O(k^2n^2 + 2^k)$.

Let C be the clique in G and let f be any 1:1 map from C to $\{1, ..., k\}$. Consider the set $S \subseteq X$ of cardinality k:

$$S = \{(u, f(u)) : u \in C\}$$

We show that every subset of S is the intersection of S and a set in F. Let S' be a subset of S. If $|S'| \geq 3$, then let $I' = \{i : (u, i) \in S'\}$, and the set $\{(u, i) : u \in V, i \in I'\}$ in F_3 intersecting S gives S'. If $|S'| = 2$, then $S' = \{(u_1, i_1), (u_2, i_2)\}$. Since C is a clique in G, $[u_1, u_2]$ is an edge in G, so S' is a set in F_2 whose intersection with S gives S'. if $|S'| = 1$ then $S' = \{(u, i)\}$ is a set in F_1 whose intersection with S gives S'. Finally, if $|S'| = 0$ then $S' = \phi$ and the empty set ϕ in F_0 intersecting S gives S'.

On the other hand, suppose S is a k-element subset of X, such that every subset S' of S is an intersection of S and some set W in F. We will call such a set W in F the "witness" of S' in F. Consider any subset S' of S with at least 3 elements, since each of the sets in $F_0 \cup F_1 \cup F_2$ contains fewer than 3 elements, the witness of S' must be in the family F_3. Since the set S has $\sum_{i=3}^{k}\binom{k}{i}$ subsets of at least 3 elements, and the family F_3 has exactly $\sum_{i=3}^{k}\binom{k}{i}$ sets, every set in F_3 is a witness of some subset of at least 3 elements in S. Therefore, for each subset of at most 2 elements in S, the witness must be in $F_0 \cup F_1 \cup F_2$. For each subset $S' = \{(u_1, i_1), (u_2, i_2)\}$ of 2 elements in S, since each set in $F_0 \cup F_1$ contains at most 1 element in S, the witness of S' must be in F_2, therefore we must have $u_1 \neq u_2$ and $[u_1, u_2]$ is an edge in G. In consequence, if we let $C = \{u : (u, i) \in S\}$, then C must be a clique of k vertices in G.

This verifies that the graph G has a clique of k vertices if and only if there is a set S of k elements such that every subset of S is an intersection of S with a set in the family F. This presents an fpt$_l$-reduction from CLIQUE to V-C DIMENSION-PARA, which, plus Theorem III.20 and Theorem III.25, proves the current theorem. □

Theorem III.32 TOUNAMENT DOMINATING SET-PARA *cannot be solved in time* $f(k)n^{o(k)}$ *for any function* f*, unless all SNP problems are solvable in subexponential time.*

PROOF. We give the fpt$_l$-reduction from DOMINATING SET to TOUNAMENT DOMINATING SET-PARA. The fpt-reduction in [36] for proving TOUNAMENT DOMINATING SET-PARA is W[2]-complete is essentially an fpt$_l$-reduction.

Given a graph $G = (V, E)$, $|V| = n$, and an integer $k > 0$, we will construct a tournament T such that T has a dominating set of size $k + 1$ if and only if G has a dominating set of size k. The size of T is $O(2^k n)$, and it can be constructed in time polynomial in n and 2^k.

The vertex set of the tournament T is partitioned into three sets: V_A, V_B and V_C. The vertices in V_A are in $1 : 1$ correspondence with the vertices of G. Denote $V_A = \{a[u] : u \in V(G)\}$. The vertices in V_B correspond to m copies of the vertices of G. Denote $V_B = \{b[i, u] : 1 \le i \le m, u \in V(G)\}$. (The value of m will be determined.) V_C consists of just a single vertex c.

The construction of T must insure that for every pair of vertices x and y, one of the directed edge (x, y) or (y, x) is present. Let T_0 be any tournament on n vertices as a “model”. Include directed edges in T to make a copy of T_0 between the vertices of each of the n-element V_A and $V_B(i) = \{b[i, u] : u \in V(G)\}$ for $i = 1, ..., m$.

Let T_1 be a tournament on m vertices that has no dominating set of size $k + 1$. It is easy to construct such a tournament with $m = O(2^{k+1})$. Consider the vertices set of T_1 is $V(T_1) = \{1, ..., m\}$. For each directed edge $[i, j]$ in T_1 include in T an directed edge from each vertex of $V_B(i)$ to each vertex of $V_B(j)$.

The adjacency of G is represented in T in the following way: for each vertex $u \in V(G)$ include directed edges from the vertex $a[u]$ to the vertices $b[i, v]$ for every $v \in N_G[u]$ and for each i, $1 \leq i \leq m$, and from every other vertex in V_B include an directed edge to $a[u]$.

Finally, there are directed edges in T from c to every vertex in V_A and from every vertex in V_B to c. This completes the construction of the graph T. It is easy to verify that T is a tounament graph.

If there is a dominating set S of size k in G, then the corresponding vertices in V_A dominate all of the vertices in V_B. Thus together with c we have a dominating set of size $k+1$ in T.

On the other hand, suppose T has a dominating set D of size $k+1$. At least one vertex of D must belong to V_B or V_C, otherwise the vertex c is not dominated. Thus there are at most k vertices of D in V_A. Let S_A denote the corresponding vertices of G. We verify that S_A is a dominating set of the graph G. If S_A is not a dominating set in G, then let x denote some vertex of G that is not dominated. Let $D_A = D \cap V_A$, and let $D_B = D \cap V_B$. The vertices $b[i, x]$ of V_B for $1 \leq i \leq m$ are not dominated in T by the vertices of D_A. The vertices of V_B can be viewed as belonging to m copies of $V(G)$ for which we have introduced the notation $V_B(i)$, $1 \leq i \leq m$. Since $|D_B| \leq k+1$ and T_1 has no dominating set of size $k+1$, D_B cannot dominate all vertices in V_B, so there is at least one $V_B[j]$ such that no vertex in $V_B[j]$ is dominated by D_B (note that by the construction of T, if any vertex in $V_B[j]$ is dominated by D_B, then all vertices in $V_B[j]$ would be also dominated by D_B). In particular, the vertex $b[j, x]$ is not dominated by D_B. By the discussion above, the vertex $b[j, x]$ is not dominated by D_A, either. Since $b[j, x]$ is also not dominated by the vertex c (there is no edge from c to V_B), we derive the contradiction that $b[j, x]$ is not dominated at all, and the set D would not be a dominating set for T. This contradiction shows that S_A must be a dominating set of the graph G. Note that $|S_A| \leq k$. This proves that there is a dominating set of size $k+1$ in T if and only if there is a dominating set of size k in G. The reduction is an fpt$_l$-reduction.

Based on the fpt$_l$-reduction from DOMINATING SET to TOURNAMENT DOMINATING SET, Theorem III.23 and Theorem III.25, the theorem is proved. $\square$

CHAPTER IV

LOWER BOUNDS FOR PTAS ALGORITHMS

In this chapter, we extend our techniques developed in the last chapter to derive computational lower bounds for polynomial-time approximation schemes (PTAS) for some well-known NP optimization problems, which include the computational biology problems such as DISTINGUISHING SUBSTRING SELECTION and LONGEST COMMON SUBSEQUENCE, and the problems in the class LOGNP.

A. Our Theorem

We prove a general theorem for deriving lower bounds for PTAS algorithms of NP optimization problems.

Lemma IV.1 *If an NP optimization problem Q has a PTAS algorithm of running time $f(1/\epsilon)n^{o(1/\epsilon)}$ for a recursive function f, then the parameterized version of Q can be solved in time $f(2k)n^{o(k)}$.*

PROOF. We consider the case that $Q = (I_Q, S_Q, f_Q, opt_Q)$ is a maximization problem.

From the PTAS algorithm A_Q for Q, we provide the parameterized algorithm $A_{\geq}$ shown in Fig. 2 for the parameterized version $Q_{\geq}$ of Q.

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

Algorithm $A_{\geq}$**:**

Input: An instance (x, k) of $Q_{\geq}$.

Output: If $opt_Q(x) \geq k$, then Output "yes"; otherwise Output "no".

begin

1. On the 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 $S_Q(x)$.

2. **If** $f_Q(x, y) \geq k$, then **return** "yes"; otherwise **return** "no".

end

Fig. 2. Algorithm $A_{\geq}$.

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 is bounded by $f(1/\epsilon)n^{o(1/\epsilon)} = f(2k)n^{o(k)}$. Thus, the problem $Q_{\geq}$ is solvable in time $f(2k)n^{o(k)}$.

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

By Lemma IV.1, we have

Theorem IV.2 *Let Q be an NP optimization problem. If the parameterized version of Q has no algorithm of time $f(k)n^{o(k)}$, then Q has no PTAS of running time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f.*

We will demonstrate the applications of Theorem IV.2 in the following sections.

B. The DSSP Problem*

Recently, the problem DISTINGUISHING SUBSTRING SELECTION has drawn a lot of attention because of its applications in computational biology such as in drug generic design [31].

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.

*Part of the data reported in this section is reprinted with permission from "Linear FPT reductions and computational lower bounds" by J. Chen, X. Huang, I. Kanj, and G. Xia, 2004, *Proceedings of the 36th ACM Symposium on Theory of Computing* (SOTC 2004), pp. 212-221, Copyright 2004 by ACM.

The DSSP problem is NP-hard [46]. Recently, Deng et al. [30] (see also [31]) 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 [30] called their algorithm a "PTAS" for the DSSP problem. Strictly speaking, neither the problem DSSP nor the algorithm in [30] 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 [30] a "PTAS-[30]" for DSSP.

1. Standard Definitions of DSSP and Its PTAS

Since our lower bound techniques for PTAS given in Theorem IV.2 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-[30] for DSSP as given in [30]. Using the systematical methods described above, we then prove that the parameterized version of DSSP-OPT is $W_l[1]$-hard, which, by Theorem III.18 and Theorem IV.2, 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-[30] algorithm for the DSSP problem.

Definition 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-[30] for DSSP. Since the PTAS-[30] 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 IV.3 *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-[30] 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-[30] 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 IV.3 shows that a PTAS-[30] for the problem DSSP is also a PTAS in the standard definition for the optimization problem DSSP-OPT.

2. PTAS Lower Bound for DSSP

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

Lemma IV.4 *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 III.23).

Let (G, k) be an instance of the DOMINATING SET problem. Suppose that the graph G has n vertices $v_1, \ldots, v_n$. 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 $v_{i_1}, \ldots, v_{i_r}$, 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 [37]. Therefore, the proof of Lemma IV.4 actually shows that the DSSP-OPT$_\geq$ problem is $W[2]$-hard. This improves the result in [46], which proved that the problem is $W[1]$-hard.

From Lemma IV.4, Theorem III.18 and Theorem IV.2, we get immediately

Theorem IV.5 *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 IV.3, a PTAS-[30] 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 IV.5 also implies that any PTAS-[30] for DSSP cannot run in time $f(1/\epsilon)m^{o(1/\epsilon)}$ for any function f. Thus essentially, no PTAS-[30] 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 IV.5 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 [5], 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 [5], 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.

C. The LCS Problem

The LONGEST COMMON SUBSEQUENCE (LCS) problem is a well-known optimization problem because of its applications ([60]). The fixed alphabet versions of the problem is of particular interest considering the importance of sequence comparison (e.g. multiple sequence alignment) in the fixed size alphabet world of DNA and protein sequences. (Note that in computational biology, DNA sequences are in a four-letter alphabet, and protein sequences are in a twenty-letter alphabet).

A string s is a *subsequence* of a string s' if s can be obtained from s' by deleting some characters in s'. For example, "ac" is a subsequence of "atcgt". Given a set of strings over an alphabet Σ, the LONGEST COMMON SUBSEQUENCE problem is to find a common subsequence that has maximum length. The alphabet Σ may be of fixed size or of unbounded size.

In [10, 11, 49, 70] several parameterized versions of the LCS problem are discussed. The following are four parameterized versions of the problem.

The **LCS-**k problem:

Instance: a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ, and an integer $\lambda > 0$, where the alphabet Σ is of unbounded size.

Parameter: k.

Question: is there a string $s \in \Sigma^*$ of length λ, which is a subsequence of each string in S?

The **FLCS-**k problem:

Instance: a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ, and an integer $\lambda > 0$, where the alphabet Σ is of fixed size.

Parameter: k.

Question: is there a string $s \in \Sigma^*$ of length λ, which is a subsequence of each string in S?

The **LCS-**λ problem:

Instance: a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ, and an integer $\lambda > 0$, where the alphabet Σ is of unbounded size.

Parameter: λ.

Question: is there a string $s \in \Sigma^*$ of length λ, which is a subsequence of each string in S?

The **FLCS-**λ problem:

Instance: a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ, and an integer $\lambda > 0$, where the alphabet Σ is of fixed size.

Parameter: λ.

Question: is there a string $s \in \Sigma^*$ of length λ, which is a subsequence of each string in S?

The following results on the parameterized complexity of these parameterized problems are known:

- The LCS-k problem is W[t]-hard for $t \geq 1$ [11].
- The FLCS-k problem is W[1]-hard [70].
- The LCS-λ problem is W[2]-hard [11].
- The FLCS-λ problem is in FPT [70].

In particular, we are interested in the FLCS-k problem and the LCS-λ problem, which we discuss in the following sections.

1. FLCS-k

In [70], the FLCS-k problem is proved to be $W[1]$-hard. Unless $W[1] =$ FPT, for the FLCS-k problem, the $W[1]$-hardness result rules out the existence of algorithms of time $f(k)n^{O(1)}$ for any function f, where k is the number of strings. In the conclusion of [70], the author pointed out that the $W[1]$-hardness of FLCS-k "does not mean that there are no algorithms with much better asymptotic time-complexity than the known $O(n^k)$ algorithms based on dynamic programming, e.g. algorithms with running time $n^{\sqrt{k}}$ are not deemed impossible."

However, we prove:

Theorem IV.6 *The FLCS-k problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. The proof is based on the fpt_l-reduction from CLIQUE to the FLCS-k problem. Based on the fpt_l-reduction, Theorem III.20 and Theorem III.25, the theorem is proved.

The fpt-reduction from CLIQUE to the FLCS-k problem in [70] for proving the FLCS-k problem is $W[1]$-hard is essentially an fpt_l-reduction.

A problem called PARTITIONED CLIQUE is first introduced:

> PARTITIONED CLIQUE: given a graph $G = (V, E)$ and a partition of V into k sets of equal sizes, $\{U_1, U_2, ..., U_k\}$, where $k > 0$, is there a clique of size k, such that there is exactly one vertex from each of the k sets?

We prove that PARTITIONED CLIQUE is $W_l[1]$-hard by an fpt_l-reduction from CLIQUE. Given an instance of the CLIQUE problem $(G = (V, E), k)$, where $V = \{v_1, v_2, ..., v_n\}$, an instance of the PARTITIONED CLIQUE problem $(G' = (V', E'), k, U)$, where $U = \{U'_1, U'_2, ..., U'_k\})$, is built as follows. Every set $U'_j = \{u^j_1, ..., u^j_n\}$ consists of n vertices. A vertex $u^j_i \in U'_j$ corresponds to vertex $v_i \in V$. There is an edge $(u^i_x, u^j_y) \in E'$ if and only if $(v_x, v_y) \in E$.

We show that G has a clique of size k if and only if G' has a partitioned clique of size k. If G has a clique C of size k. we can assign every vertex from C to the corresponding vertex in a different set U'_i. By the construction, these vertices form a partitioned clique in G'. On the other hand, if we are given a partitioned clique C' in G', then each vertex in C' corresponds to a different vertex in G (two vertices that correspond to the same vertex in G are not adjacent in G'), and those vertices build a clique in G by the construction. Therefore, there is an fpt_l-reduction from CLIQUE to PARTITIONED CLIQUE.

Now we present the fpt_l-reduction from PARTITIONED CLIQUE to the FLCS-k

problem. Given an instance of PARTITIONED CLIQUE (G, k, U), where $G = (V, E)$, $U = \{U_1, U_2, ..., U_k\}$, an instance of the FLCS-k problem $(S = \{s_1, s_2, ..., s_k, s_t\}, \lambda)$ is built, where there are $k+1$ strings, and λ is the length of the common subsequence. The alphabet is $\{0, 1\}$.

Let $n = |V|$, $m = |U_i| = n/k$. Define the following strings from which the instance of the FLCS-k problem is constructed.

$I = 1^{7n^3}$;

$O = 0^{7n^3}$;

$\varepsilon(u \in V, v \in V) = II$, if $(u = v)$ or $(u \in U_i,\ v \in U_j) \in E : i \neq j$; otherwise, $\varepsilon(u \in V, v \in V) = I0I$;

$\nu(u \in V) = \Pi_{j=1}^{n} \varepsilon(u, v_j)$;

$B_i = \nu(v_1^j) \Pi_{j=2}^{m} O \varepsilon(v_j^i)$;

B_i' represents the string obtained from B_i by replacing all occurrences of II with $I0I$, and vice visa.

$\tau_{I0I} = (I0I)^n$;

$\tau_{II} = (II)^n$;

$\tau = (\tau_{I0I} O)^{m-1} \tau_{I0I}$;

τ' represents the string obtained from τ by replacing all occurrences of II with $I0I$, and vice visa.

The instance of the FLCS-k problem $(S = \{s_1, s_2, ..., s_k, s_t\}, \lambda)$ is

$s_i = (B_i' O)^{2n+2n^2} B_i'$;

$s_t = \tau_{IOI} (O \tau')^{2n+2n^2}$;

$\lambda = |s_t| + (1 + 2n + 2n^2)(n - k)$;

The following are proved in [70]:

Fact 1 *If G has a partitioned clique of size k, then there is a string s_λ of length λ*

that is a common subsequence for $S = \{s_1, s_2, ..., s_k, s_t\}$.

Fact 2 *If* G *has no partitioned cliques of size* k*, then the longest common subsequence for* $S = \{s_1, s_2, ..., s_k, s_t\}$ *is less than* λ.

That is, G has a partitioned clique of size k if and only if there is a string s_λ of length λ that is a subsequence of all the $k+1$ strings in S. We have an fpt$_l$-reduction from PARTITIONED CLIQUE to the FLCS-k problem.

From the transitivity of fpt$_l$-reduction, we have an fpt$_l$-reduction from CLIQUE to the FLCS-k problem. □

We define an optimization problem FLCS-k_{opt} and its corresponding parameterized problem FLCS'-k.

The **FLCS-**k_{opt} problem:

given a set $S = \{s_1, s_2, ..., s_l\}$ of strings over a fixed alphabet Σ, and an integer $\lambda > 0$, try to find a string $s \in \Sigma^*$ of length λ maximizing the size of a subset S' of S, such that s is a common subsequence of all the strings in S'.

By our definition, the parameterized version of the optimization problem FLCS-k_{opt} is

The **FLCS'-**k problem:

Instance: given a set $S = \{s_1, s_2, ..., s_l\}$ of strings over a fixed alphabet Σ, and an integer $\lambda > 0$.

Parameter: an integer k, $0 < k \le l$.

Question: is there a string $s \in \Sigma^*$ of length λ such that s is a common subsequence of at least k strings in the set S?

From the definitions of the two parameterized problems FLCS-k and FLCS'-k, we can see that FLCS-k is a special case of FLCS'-k. There is a trivial fpt$_l$-reduction from FLCS-k to FLCS'-k: given an instance I_1 of FLCS-k, $I_1 = (S_1 = \{s_1, s_2, ..., s_k\}, \lambda$ and the parameter k), we build an instance I_2 of FLCS'-k, $I_2 = (S_2 = \{s_1, s_2, ..., s_k\}, \lambda$ and the parameter k), which asks if there is a string $s \in \Sigma^*$ of length λ that is a common subsequence of at least k strings (i.e., all strings) in the set S_2. Obviously, the instance I_2 is a yes-instance for the problem FLCS'-k if and only if the instance I_1 is a yes-instance for the problem FLCS-k, .

By the above fpt$_l$-reduction, Theorem IV.6 and Theorem III.25, we have

Lemma IV.7 *The FLCS'-k problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Therefore, by Lemma IV.7 and Theorem IV.2, we have

Theorem IV.8 *The FLCS-k_{opt} problem has no PTAS of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

2. LCS-λ

The LCS-λ problem is proved to be $W[2]$-hard in [10, 11]. Therefore, unless $W[2] =$ FPT, for the LCS-λ problem, there is no algorithm of time $f(\lambda)n^{O(1)}$ for any function f. We prove

Theorem IV.9 *The LCS-λ problem has no algorithm of time $f(\lambda)n^{o(\lambda)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

PROOF. We first give an fpt$_l$-reduction from DOMINATING SET to the LCS-λ problem. Based on the fpt$_l$-reduction, Theorem III.23 and Theorem III.25, the theorem is proved.

The fpt-reduction from DOMINATING SET to the LCS-λ problem in [11] for proving the LCS-λ problem is $W[2]$-hard is essentially an fpt$_l$-reduction.

Given a graph $G = (V, E)$, $|V| = n$, and a parameter λ, and suppose an ascending order of the vertices $\{u_1, u_2, ..., u_n\}$ of G, we will construct a set S of strings such that they have a common subsequence of length λ if and only if G has a dominating set of size λ. The alphabet is $\Sigma = \{a[i, j] : 1 \leq i \leq \lambda, 1 \leq j \leq n\}$. We use the notations: $\Sigma_i = \{a[i, j] : 1 \leq j \leq n\}$, $\Sigma[t, u] = \{a[i, j] : (i \neq t) \text{ or } (i = t \text{ and } j \in N[u])\}$.

If $\Gamma \subseteq \Sigma$, let $(\uparrow \Gamma)$ be the string of length $|\Gamma|$ which consists of one occurrence of each symbol in Γ in ascending order, and let $(\downarrow \Gamma)$ be the string of length $|\Gamma|$ which consists of one occurrence of each symbol in Γ in descending order.

The set S consists of the following strings.

Control strings:

$X_1 = \Pi_{i=1}^{\lambda}(\uparrow \Sigma_i)$,

$X_2 = \Pi_{i=1}^{\lambda}(\downarrow \Sigma_i)$.

Check strings: For $u = 1, ..., n$:

$X_u = \Pi_{i=1}^{\lambda}(\uparrow \Sigma[i, u])$,

We observe that any sequence C of length λ that is a common subsequence of both control strings must consist of exactly one symbol from each Σ_i in ascending order. For such a sequence C we may associate the set V_c of vertices represented by C: if $C = a[1, u_1]...a[\lambda, u_\lambda]$, then $V_c = \{u_i : 1 \leq i \leq \lambda\} = \{x : \exists i \; a[i, x] \in C\}$.

We will prove that if C is also a subsequence of the check strings $\{X_u\}$, then V_c is a dominating set in G. Let $u \in V(G)$ and fix a substring C_u of X_u, with $C_u = C$. We have the fact:

Fact 3 ([11]) *For some index j, $1 \leq j \leq \lambda$, the symbol $a[j, u_j]$ occurs in the $(\uparrow \Sigma[j, u])$ portion of X_u, thus $u_j \in N[u]$ by the definition of $\Sigma[j, u]$.*

By Fact 3, if C is a subsequence of the control and check strings, then every vertex of G has a neighbor in V_c, that is, V_c is a dominating set in G.

On the other hand, if $D = \{u_1, .., u_\lambda\}$ is a dominating set in G with $u_1 < ... < u_\lambda$, then the sequence $C = a[1, u_1]...a[\lambda, u_\lambda]$ is easily seen to be a common subsequence of the strings in S.

The reduction from DOMINATING SET to LCS-λ is an fpt$_l$-reduction. $\square$

Formally, we give the definition of the optimization problem LCS-λ_{opt}.

The **LCS-**λ_{opt} problem:

given a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ of unbounded size, try to find a string $s \in \Sigma^*$ of maximum length such that s is a common subsequence of all the strings in S.

By our definition, the parameterized version of the optimization problem LCS-λ_{opt} is

The **LCS'-**λ problem:

Instance: given a set $S = \{s_1, s_2, ..., s_k\}$ of strings over an alphabet Σ of unbounded size.

Parameter: an integer $\lambda > 0$.

Question: is there a string $s \in \Sigma^*$ of length at least λ such that s is a common subsequence of all strings in the set S?

Since that there is a string s of length at least λ such that s is a common subsequence of all strings in S is equivalent to that there is a string s of length exactly λ such that s is a common subsequence of all strings in S, the two problems LCS-λ and LCS'-λ are equivalent. By Theorem IV.9, the problem LCS'-λ has no algorithm of time $f(\lambda)n^{o(\lambda)}$ for any function f, unless all SNP problems are solvable in subexponential time. This result plus Theorem IV.2 gives us the following theorem:

Theorem IV.10 *The LCS-λ_{opt} problem has no PTAS of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

In [55], the authors showed that the LCS-λ_{opt} problem is inherently hard to approximate in the worst case. In particular, they proved that there exists a constant $\delta > 0$ such that, the LCS-λ_{opt} has no polynomial time approximation algorithm with performance ratio n^{δ}, unless P = NP. It is obvious to see that this lower bound holds only when the objective function value λ is larger than n^d for a constant $d > 0$. In particular, the lower bound result in [55] does not apply to the case when the value of λ is small. For example, in case $\lambda = n^{\delta}$, a trivial common subsequence of length one is a ratio-n^{δ} approximation solution. This implies that for the LCS problem, when the length λ of the common subsequence is a small function of n, no strong lower bound result as that of [55] has been derived.

On the other hand, our lower bound result in Theorem IV.10 for the LCS problem can be applied when the length of the common subsequence λ is any small function of the length n of each string.

D. The LOGNP Problems

In the previous chapter we have derived computational lower bounds for the parameterized LOGNP problems. Here we discuss the optimization versions of the decision problems in the class LOGNP.

1. Rich Hypergraph Cover, Tournament Dominating Set and V-C Dimension

For the decision problem RICH HYPERGRAPH COVER, we define the RICH HY-

PERGRAPH COVER-OPT problem as its optimization problem.

> RICH HYPERGRAPH COVER-OPT: given a hypergraph $H = (V, E)$, where $|V| = n$ and all edges of size at least $n/2$, try to find a minimum vertex cover for H.

By our definition, the parameterized version of the optimization problem RICH HYPERGRAPH COVER-OPT is

> RICH HYPERGRAPH COVER-PARA': given a hypergraph $H = (V, E)$, where $|V| = n$ and all edges of size at least $n/2$, and a parameter k, where $k \leq \log n$, is there a vertex cover for H of size at most k?

For the decision problem TOURNAMENT DOMINATING SET, we define the TOURNAMENT DOMINATING SET-OPT problem as its optimization problem.

> TOURNAMENT DOMINATING SET-OPT: given a tournament graph G, try to find a minimum dominating set for the graph G.

By our definition, the parameterized version of the optimization problem TOURNAMENT DOMINATING SET-OPT is

> TOURNAMENT DOMINATING SET-PARA': given a tournament graph G, and a parameter k, is there a dominating set of size at most k for the graph G?

For the decision problem V-C DIMENSION, we define the V-C DIMENSION-OPT problem as its optimization problem.

> V-C DIMENSION-OPT: given a family C of subsets of a universe U, try to maximize the size of the subset S of U such that for each subset T of S, there is a set $C_T \in C$ satisfying $S \cap C_T = T$.

By our definition, the parameterized version of the optimization problem V-C DIMENSION-OPT is

> V-C DIMENSION-PARA’: given a family C of subsets of a universe U, and a parameter k, is there a subset S of U such that for each subset T of S, there is a set $C_T \in C$ satisfying $S \cap C_T = T$, and the size of S is at least k?

We can verify that the above parameterized problems: RICH HYPERGRAPH COVER-PARA’, TOURNAMENT DOMINATING SET-PARA’ and V-C DIMENSION-PARA’, are equivalent to the parameterized problems: RICH HYPERGRAPH COVER-PARA, TOURNAMENT DOMINATING SET-PARA and V-C DIMENSION-PARA, which we described in Chapter III. By Theorem III.29, III.31, and III.32, we have

Lemma IV.11 *The parameterized problems:* RICH HYPERGRAPH COVER-PARA’, TOURNAMENT DOMINATING SET-PARA’, *and* V-C DIMENSION-PARA’, *have no algorithms of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Based on Lemma IV.11 and Theorem IV.2, we have the following lower bound results for the optimization problems.

Theorem IV.12 *The optimization problems:* RICH HYPERGRAPH COVER-OPT, TOURNAMENT DOMINATING SET-OPT, *and* V-C DIMENSION-OPT, *have no PTAS algorithms of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

In particular, our inapproximability result for the V-C DIMENSION-OPT problem in Theorem IV.12 answers the open problem posed in the literature [16].

2. LOG Hypergraph Cover, LOG Adjustment, and LOG Dominating Set

For the decision problem LOG HYPERGRAPH COVER, we define the LOG HYPERGRAPH COVER-OPT problem as its optimization problem.

> LOG HYPERGRAPH COVER-OPT: given a hypergraph $H = (V, E)$ and a subset V_c of V, where $|V_c| = \log n$ and V_c is a cover of H, try to find a minimum cover of H.

By our definition, the parameterized version of the optimization problem LOG HYPERGRAPH COVER-OPT is

> The LOG HYPERGRAPH COVER-PARA' problem: given a hypergraph $H = (V, E)$, a subset V_c of V, where $|V_c| = \log n$ and V_c is a cover of H, and a parameter k, is there a cover of H of size at most k?

We show that the RICH HYPERGRAPH COVER-PARA' problem is fpt_l-reducible to the LOG HYPERGRAPH COVER-PARA' problem. Given an instance I_1 of RICH HYPERGRAPH COVER-PARA', $I_1 = (H = (V, E), k)$, where $|V| = n$, each hyperedge of H contains at least $n/2$ vertices, and $k \leq \log n$, we build an instance $I_2 = (H = (V, E), V_c, k)$ as follows. First let $V_c = \emptyset$. Since all hyperedges of H contain at least $n/2$ vertices, there exists such a vertex $v_1 \in V$ that is contained in at least half of the hyperedges. We can check each vertex $v \in V$ and find such a vertex v_1. We add v_1 into the set V_c (v_1 covers half of the hyperedges). In the same way, we can find another vertex v_2 that is contained in at least half of the remaining hyperedges. We add v_2 into the set V_c. Keep doing this until we have a vertex set $V_c = \{v_1, v_2, ..., v_{\log n}\}$ that covers all the hyperedges of H. $I_2 = (H, V_c, k)$ is an instance of the LOG HYPERGRAPH COVER-PARA' problem. Obviously, the instance I_1 is a yes-instance of RICH HYPERGRAPH COVER-PARA' if and only if the instance I_2 is a yes-instance of LOG

HYPERGRAPH COVER-PARA'. The reduction from RICH HYPERGRAPH COVER-PARA' to LOG HYPERGRAPH COVER-PARA' is an fpt_l-reduction.

By the above fpt_l-reduction, Lemma IV.11 and Theorem III.25, we have

Lemma IV.13 *The* LOG HYPERGRAPH COVER-PARA' *problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Therefore, by Lemma IV.13 and Theorem IV.2, we have the following theorem.

Theorem IV.14 *The* LOG HYPERGRAPH COVER-OPT *problem has no PTAS algorithm of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

For the decision problem LOG ADJUSTMENT, we define the LOG ADJUSTMENT-OPT problem as its optimization problem.

> LOG ADJUSTMENT-OPT: given a Boolean expression F in conjunctive normal form with n variables, and a truth assignment T, and also a satisfying truth assignment T' whose Hamming distance from T is $\log n$, try to find a satisfying truth assignment with the minimum Hamming distance from T.

By our definition, the parameterized version of the optimization problem LOG ADJUSTMENT-OPT is

> The LOG ADJUSTMENT-PARA' problem: given a Boolean expression F in conjunctive normal form with n variables, a truth assignment T, a satisfying truth assignment T' whose Hamming distance from T is $\log n$, and a parameter k, is there a satisfying truth assignment whose Hamming distance from T is at most k?

We show that the RICH HYPERGRAPH COVER-PARA' problem is fpt_l-reducible to the LOG ADJUSTMENT-PARA' problem. Given an instance I_1 of RICH HYPERGRAPH COVER-PARA', $I_1 = (H = (V, E), k)$, where $|V| = n$, each hyperedge of H contains at least $n/2$ vertices, and $k \leq \log n$, we build an instance $I_2 = (F, T, T', k)$ as follows. F is a conjunctive normal form with n positive input variables $\{v_1, v_2, ..., v_n\}$. The n positive input variables represent the n vertices of H. Each clause of F, which corresponds to an edge e of the hypergraph H, is a disjunction of all the variables that represent the vertices of the edge e. We assign all variables FALSE as the default truth assignment T. From our discussion of the fpt$_l$-reduction from RICH HYPERGRAPH COVER-PARA' to LOG HYPERGRAPH-PARA', we know that for the hypergraph H, we can find a vertex set $V_c = \{v_1, v_2, ..., v_{\log n}\}$ that covers all the hyperedges of H. We assign all variables that correspond to the vertices in V_c TRUE and all other variables FALSE as the truth assignment T'. T' is a satisfying truth assignment whose Hamming distance from T is $\log n$. $I_2 = (F, T, T', k)$ is an instance of the LOG ADJUSTMENT-PARA' problem.

We show that the instance I_1 is a yes-instance of RICH HYPERGRAPH COVER-PARA' if and only if the instance I_2 is a yes-instance of LOG ADJUSTMENT-PARA': suppose there is a cover C of size k for the hypergraph H. There are k variables in F corresponding to the k vertices of C. We assign the k variables TRUE and get a truth assignment T''. From the construction of F, T'' is a satisfying truth assignment and its Hamming distance from T is k. On the other hand, suppose there is a satisfying truth assignment T'' whose Hamming distance from T is k (that is, in T'' there are k variables being assigned TRUE). In H, the k vertices that correspond to the k variables cover all the hyperedges of H.

The reduction from RICH HYPERGRAPH COVER-PARA' to LOG ADJUSTMENT-PARA' is an fpt_l-reduction.

By the above fpt_l-reduction, Lemma IV.11 and Theorem III.25, we have

Lemma IV.15 *The* LOG ADJUSTMENT-PARA' *problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Therefore, by Lemma IV.15 and Theorem IV.2, we have the following theorem.

Theorem IV.16 *The* LOG ADJUSTMENT-OPT *problem has no PTAS algorithm of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

For the decision problem LOG DOMINATING SET, we define the LOG DOMINATING SET-OPT problem as its optimization problem.

> LOG DOMINATING SET-OPT: given a graph $G = (V, E)$ and a subset V_{DS} of V, where $|V_{DS}| = \log n$ and V_{DS} is a dominating set of G, try to find a minimum dominating set of G.

By our definition, the parameterized version of the optimization problem LOG DOMINATING SET-OPT is

> The LOG DOMINATING SET-PARA' problem: given a graph $G = (V, E)$, a subset V_{DS} of V, where $|V_{DS}| = \log n$ and V_{DS} is a dominating set of G, and a parameter k, is there a dominating set of size at most k?

We show that the LOG HYPERGRAPH COVER-PARA' problem is fpt_l-reducible to the LOG DOMINATING SET-PARA' problem. Given an instance I_1 of LOG HYPERGRAPH COVER-PARA', $I_1 = (H = (V_H, E_H), V_c, k)$, where H is a hypergraph, $|V_H| = n$, V_c is a cover of size $\log n$ for H, we build an instance $I_2 = (G = (V, E), V_{DS}, k)$, where $V = V_1 \cup V_2$, as follows. The vertex set $V_2 = V_H$ contains n vertices. There are edges between any two of the n vertices. For each hyperedge $e \in E_H$, there is a vertex $v_e \in V_1$ corresponding to e. There is an edge between a vertex $v_e \in V_1$ and a vertex $v_i \in V_2$ if and only if in H the corresponding hyperedge $e \in E_H$ contains the vertex $v_i \in V_H$. We can see that the vertex set V_1 makes an independent set and the vertex set V_2 induces a clique. Since V_c is a cover of H which has $\log n$ vertices, we can see that in G, the corresponding $\log n$ vertices consist of a dominating set V_{DS}. $I_2 = (G, V_{DS}, k)$ is an instance of the LOG DOMINATING SET-PARA' problem.

Similar to our discussion in the proof of Theorem III.19, we show that the instance I_1 is a yes-instance of LOG HYPERGRAPH COVER-PARA' if and only if the instance I_2 is a yes-instance of LOG DOMINATING SET-PARA': suppose H has a cover C of size k. Then by the construction of the graph G, the corresponding k vertices in V_2 consist of a dominating set for G. On the other hand, suppose G has a dominating set D of size k, by the discussion before Lemma III.15, we can assume that D is a subset of V_2. Since the k vertices in D dominate all the vertices in V_1, the corresponding k vertices in H cover all the hyperedges of H. That is, H has a cover of size k.

The reduction from LOG HYPERGRAPH COVER-PARA' to LOG DOMINATING SET-PARA' is an fpt_l-reduction.

By the above fpt_l-reduction, Lemma IV.13 and Theorem III.25, we have

Lemma IV.17 *The* LOG DOMINATING SET-PARA' *problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Therefore, by Lemma IV.17 and Theorem IV.2, we have the following theorem.

Theorem IV.18 *The* LOG DOMINATING SET-OPT *problem has no PTAS algorithm of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

3. $\log n$-partite Graph Clique

For the decision problem LOG CLIQUE, we define the $\log n$-PARTITE GRAPH CLIQUE problem as its optimization problem. A $\log n$-partite graph G has $\log n$ partitions of vertices with each partition n vertices.

> The $\log n$-PARTITE GRAPH CLIQUE problem: given a $\log n$-partite graph G, try to find the maximum clique of the graph G.

We can see that the size of the maximum clique of a $\log n$-partite graph is less than or equal to $\log n$.

Note that the $\log n$-PARTITE GRAPH CLIQUE problem has found applications in computational biology [76, 20].

By our definition, the parameterized version of the optimization problem $\log n$-PARTITE GRAPH CLIQUE is

> The $\log n$-PARTITE GRAPH CLIQUE-PARA problem: given a $\log n$-partite graph G and a parameter k, is there a clique of size at least k in G?

It is not difficult to show that the LOG CLIQUE-PARA problem we defined in Chapter III is fpt_l-reducible to the $\log n$-PARTITE GRAPH CLIQUE-PARA problem. Given an instance I_1 of LOG CLIQUE-PARA, $I_1 = (G = (V, E), k)$, where $V = \{v_1, v_2, ..., v_n\}$ and $k \leq \log n$, we build an instance I_2 of $\log n$-PARTITE GRAPH CLIQUE-PARA, $I_2 = (G' = (V', E'), k)$ as follows. G' has $\log n$ copies of the vertices in G. We denote $V' = \{V_1, V_2, ..., V_{\log n}\}$, where each copy V_i has n vertices $\{v_{i1}, v_{i2}, ..., v_{in}\}$, for $1 \leq i \leq \log n$. The vertex v_{ix} in G', where $1 \leq i \leq \log n$ and $1 \leq x \leq n$, corresponds to the vertex v_x in G. We build edges between two vertices v_{ix} and v_{jy} in G' if and only if $i \neq j$ and $(v_x, v_y) \in E$. We can see that the graph G' is a $\log n$-partite graph, with each copy of the vertices in G as a partition, and there are edges between vertices from different partitions.

We show that G has a clique of size k if and only if G' has a clique of size k. Suppose G has a clique $C = \{v_{c_1}, v_{c_2}, ..., v_{c_k}\}$, where each $c_i \in \{1, 2, ..., n\}$. Then from the construction of G', there is a clique $C' = \{v_{1c_1}, v_{2c_2}, ..., v_{kc_k}\}$ in G'. On the other hand, suppose there is a clique C' of size k in G', we know that all the k vertices in C' should be from different partitions and any two of them are not copies of the same vertex of G (since by the construction of G', there are no edges between copies of the same vertex of G). Then the k vertices in C' corresponds to k different vertices in G. Furthermore, since there is an edge between any two of the k vertices in C' (C' is a clique), there is an edge between any two of the corresponding k vertices in G. That is, G has a clique of size k.

The reduction from LOG CLIQUE-PARA to $\log n$-PARTITE GRAPH CLIQUE-PARA is an fpt_l-reduction.

By the above fpt_l-reduction, Theorem III.26 and Theorem III.25, we have

Lemma IV.19 *The* $\log n$-PARTITE GRAPH CLIQUE-PARA *problem has no algorithm of time $f(k)n^{o(k)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Therefore, by Lemma IV.19 and Theorem IV.2, we have the following theorem.

Theorem IV.20 *The* $\log n$-PARTITE GRAPH CLIQUE *problem has no PTAS algorithm of time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any function f, unless all SNP problems are solvable in subexponential time.*

Before ending the section, we point out that for the decision problem LOG CHORDLESS PATH, we do not have a natural optimization version.

CHAPTER V

STUDY OF EPTAS ALGORITHMS ON PLANAR GRAPHS

So far we can prove lower bound results for NP optimization problems when the parameterized versions of these problems are W[t]-hard, $t \geq 1$. In this chapter, we discuss the lower bounds for the parameterized problems that are fixed-parameter tractable.

We prove computational lower bounds on the EPTAS algorithms for some famous planar graph NP-hard optimization problems. Based on the result in [17] (Lemma 6), the parameterized versions of these optimization problems are in FPT.

A. EPTAS Lower Bound Results

Based on the outer-planarity of planar graphs, Baker [7] designed EPTAS algorithms of time $O(2^{O(1/\epsilon)}n)$ for several famous NP-hard optimization problems on planar graphs, such as PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and PLANAR DOMINATING SET, where $\epsilon > 0$ is the given error bound, and n is the number of vertices of the planar graph.

Alber et. al [3] designed parameterized algorithms of time $2^{O(\sqrt{k})}n^{O(1)}$ for the parameterized versions of the above NP-hard optimization problems on planar graphs. A lot of research has been done on these problems to try to further improve the time complexity of the parameterized algorithms. Interested readers are referred to [2, 56, 41, 42].

Cai et. al [15] proved the following lower bound result for the parameterized algorithms of these problems.

Theorem V.1 ([15]) *PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and*

PLANAR DOMINATING SET *do not have parameterized algorithms of time* $2^{o(\sqrt{k})}n^{O(1)}$, *unless all SNP problems are solvable in subexponential time.*

From Theorem V.1 and Theorem IV.2, we have

Theorem V.2 PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, *and* PLANAR DOMINATING SET *have no EPTAS of running time* $2^{o(\sqrt{1/\epsilon})}n^{O(1)}$, *where* $\epsilon > 0$ *is the given error bound, unless all SNP problems are solvable in subexponential time.*

Note that the upper bound of the EPTAS algorithms for the above problems in Baker [7] is $2^{O(1/\epsilon)}n^{O(1)}$ (also [62]). We can see that there is a gap between the upper bound and our lower bound result. To come up with new approaches to improve the upper bound of the EPTAS algorithms in [7] will be interesting research. To study this issue, we concentrate on the PLANAR VERTEX COVER problem in the next section.

B. Planar Vertex Cover and EPTAS Upper Bound

We study the EPTAS algorithm of the VERTEX COVER problem on planar graphs of degree bounded by 3, abbreviated as P-VC-3. The VERTEX COVER problem on general planar graphs is abbreviated as P-VC.

From Theorem IV.2, we get the following lemma:

Lemma V.3 *The* P-VC-3 *problem has no EPTAS of running time* $2^{o(\sqrt{1/\epsilon})}n^{O(1)}$, *where* $\epsilon > 0$ *is the given error bound, unless the* P-VC-3 *problem has a parameterized algorithm of time* $2^{o(\sqrt{k})}n^{O(1)}$.

It is well known that a planar embedding of a planar graph can be constructed in linear time [51]. We define an operation, called the *unfolding operation*, based on a planar embedding of a planar graph.

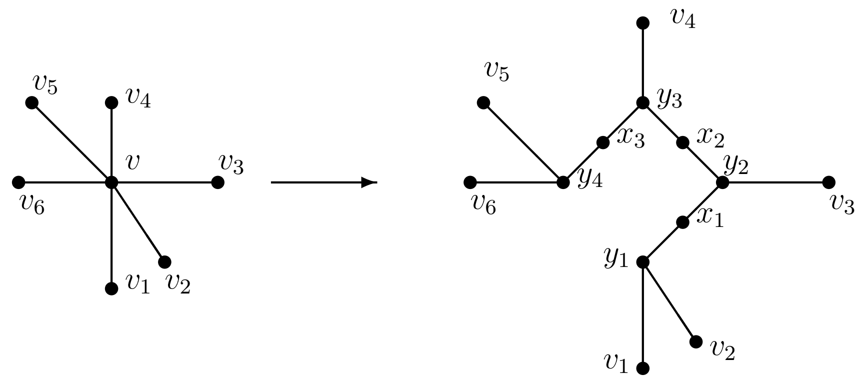

Fig. 3. Unfolding operation on the vertex v (with degree 6).

Definition Suppose that G is a planar graph with a planar embedding $\pi(G)$, and that v is a degree-d vertex in G, where $d > 3$, with neighbors v_1, v_2, …, v_d, such that when one traverses around the vertex v on the embedding $\pi(G)$, the edges incident to v are in the cyclic order $[v, v_1]$, $[v, v_2]$, …, $[v, v_d]$. The *unfolding operation* on the vertex v will do the following: remove the vertex v from $\pi(G)$, and add a path of length $2d - 5$:

$$P_v = \{y_1, x_1, y_2, x_2, \ldots, y_{d-3}, x_{d-3}, y_{d-2}\}$$

where each vertex x_i is of degree 2 and adjacent to the vertices y_i and y_{i+1}, and each vertex y_i is of degree 3 such that y_1 is adjacent to $\{v_1, v_2, x_1\}$, y_{d-2} is adjacent to $\{v_{d-1}, v_d, x_{d-3}\}$, and y_i is adjacent to $\{v_{i+1}, x_{i-1}, x_i\}$, for $2 \leq i \leq (d - 3)$.

As an example, please refer to the unfolding operation on the vertex v of degree 6 shown in Fig. 3. Note that the unfolding operation does not change the planarity of a graph: the path P_v can be drawn on a small disc on which the vertex v was embedded in $\pi(G)$, and the edges from the vertices v_1, …, v_d to the path P_v can be drawn on the plane without edge crossing.

Suppose we are given a planar graph $G_1 = (V_1, E_1)$, $V_1 = V_{\leq 3} \cup V_{>3}$, where $V_{\leq 3}$ is the set of vertices whose degree is less than or equal to 3, $V_{>3}$ is the set of vertices whose degree is greater than 3. We apply the unfolding operation on a vertex $v \in V_{>3}$.

We get a new planar graph $G_2 = (V_2, E_2)$, where G_2 has one fewer vertex of degree larger than 3, compared with G_1.

We first consider a vertex cover C_2 of the graph G_2.

- Suppose for some i, $1 \leq i \leq d-3$, the three vertices x_i, y_i, and y_{i+1} are all in C_2. Then we simply remove x_i from C_2. It is obvious that $C_2 - \{x_i\}$ is still a vertex cover of G_2, with one fewer vertex compared with C_2. Call this operation *clean-one*.

- Suppose for some i, $1 \leq i \leq d-3$, exactly two of the three vertices x_i, y_i, and y_{i+1} are in C_2. If one of these two vertices is x_i, then we can replace the two vertices by y_i and y_{i+1}, resulting in a new vertex cover of the same size. Call this operation *clean-two*.

Note that at least one of the three vertices x_i, y_i, and y_{i+1} must be in the vertex cover C_2 in order to cover the edges $[x_i, y_i]$ and $[x_i, y_{i+1}]$. Therefore, besides the above cases, the only remaining case is that for the three vertices x_i, y_i, and y_{i+1}, only one of them is in C_2. In this case, this vertex in C_2 must be x_i.

In the following discussion, *cleaning* a vertex cover C_2 means that we apply the processing of clean-one and clean-two on C_2. After the cleaning process, we say that the vertex cover C_2 is *clean*. By the above discussion, in a clean vertex cover C_2 of the graph G_2, we have

Claim 1 *Either all $d-3$ vertices x_i, $1 \leq i \leq d-3$, are in C_2 and none of the $d-2$ vertices y_j, $1 \leq j \leq d-2$, is in C_2; or all $d-2$ vertices y_j, $1 \leq j \leq d-2$, are in C_2 and none of the $d-3$ vertices x_i, $1 \leq i \leq d-3$, is in C_2.*

Let C_1 be any vertex cover of the graph G_1 such that C_1 has k_1 vertices. If $v \in C_1$ (so v covers the d edges $[v, v_1]$, $\ldots$, $[v, v_d]$ in G), then by replacing v in C_1 by the $d-2$ vertices y_1, y_2, …, y_{d-2} in G_2, we obviously get a clean vertex cover C_2 for the graph G_2. The vertex cover C_2 has $k_1 + (d-3)$ vertices. On the other hand, if v is not in C_1 (so the edges $[v, v_1]$, …, $[v, v_d]$ must be covered by the vertices v_1, …, v_d in C_1), then by adding the $d-3$ vertices x_1, x_2, …, x_{d-3} to C_1, we get a clean vertex cover C_2 for the graph G_2 and C_2 contains $k_1+(d-3)$ vertices. In conclusion, from a vertex cover of k_1 vertices for the graph G_1, we can always construct a (clean) vertex cover of $k_1 + (d-3)$ vertices for the graph G_2.

Conversely, suppose that we are given a clean vertex cover C_2 of the graph G_2, where C_2 has k_2 vertices. If C_2 contains the $d-2$ vertices y_1, y_2, …, y_{d-2}, then replacing the $d-2$ vertices y_1, y_2, …, y_{d-2} in C_2 by a single vertex v gives a vertex cover of $k_2 - (d-3)$ vertices for the graph G_1. On the other hand, if C_2 contains the $d-3$ vertices x_1, x_2, …, x_{d-3}, then removing these $d-3$ vertices from C_2 gives a vertex cover of $k_2 - (d-3)$ vertices for the graph G_1. In conclusion, from a vertex cover of k_2 vertices for the graph G_2, we can always construct a vertex cover of $k_2 - (d-3)$ vertices for the graph G_1.

Now suppose that the set of vertices of degree larger than 3 in the graph G_1 is $V_{>3} = \{u_1, u_2, \ldots, u_r\}$. Denote by $deg(u)$ the degree of the vertex u. Inductively, suppose that the graph G_{i+1} is obtained from the graph G_i by unfolding the vertex u_i, for $1 \leq i \leq r$. Note that the graph G_r has its degree bounded by 3, and we say that the graph G_r is obtained from the graph G_1 by *unfolding all vertices of degree larger than 3*. Let C_1 be a vertex cover for the graph G_1 with $|C_1| = k_1$. By the above discussion, we can construct from C_1 a vertex cover C_2 of $k_1 + (deg(u_1) - 3)$ vertices for the graph G_2; then from C_2, we can construct a vertex cover C_3 of $k_1 + (deg(u_1) - 3) + (deg(u_2) - 3)$ vertices for the graph G_3, ……, and finally we construct a vertex cover C_r of $k_1 + \sum_{i=1}^{r}(deg(u_i) - 3)$ vertices for the graph G_r.

On the other hand, let C_r be a vertex cover of k_r vertices for the graph G_r. First we clean C_r to get a clean vertex cover C'_r for G_r. Since cleaning does not increase the size of the vertex cover, we have $|C'_r| \leq |C_r| = k_r$. Now by the above discussion, we can get a vertex cover C_{r-1} of $|C'_r| - (deg(u_r) - 3) \leq k_r - (deg(u_r) - 3)$ vertices for the graph G_{r-1}. Cleaning the vertex cover C_{r-1} gives us a clean vertex cover C'_{r-1} for the graph G_{r-1}, and by the above processing we can get a vertex cover C_{r-2} of $|C'_{r-1}| - (deg(u_{r-1}) - 3) \leq k_r - (deg(u_r) - 3) - (deg(u_{r-1}) - 3)$ vertices for the graph G_{r-2}, ..., finally, we will construct a vertex cover of at most $k_r - \sum_{i=1}^{r}(deg(u_i) - 3)$ vertices for the graph G_1.

In particular, the above discussion enables us to derive a relation between the minimum vertex covers for the graphs G_1 and G_r. Let k_1 and k_r be the sizes of minimum vertex covers of the graph G_1 and G_r, respectively. By the above discussion, from a minimum vertex cover for the graph G_1, we can construct a vertex cover of $k_1 + \sum_{i=1}^{r}(deg(u_i) - 3)$ vertices for the graph G_r. Therefore, $k_1 + \sum_{i=1}^{r}(deg(u_i) - 3) \geq k_r$. On the other hand, from a minimum vertex cover of the graph G_r, we can construct a vertex cover of no more than $k_r - \sum_{i=1}^{r}(deg(u_i) - 3)$ vertices for the graph G_1, thus $k_r - \sum_{i=1}^{r}(deg(u_i) - 3) \geq k_1$. Combining these two relations, we get $k_1 + \sum_{i=1}^{r}(deg(u_i) - 3) = k_r$.

Summarizing the above discussion, we get the following:

Claim 2 *Let G_1 be a graph in which the set of vertices of degree larger than 3 is $V_{>3}$. Let G_r be a graph obtained by unfolding all vertices of degree larger than 3 in G_1. Then from a vertex cover C_1 for the graph G_1, we can construct in polynomial time a vertex cover of $|C_1| + \sum_{u \in V_{>3}}(deg(u) - 3)$ vertices for the graph G_r; and from a vertex cover C_r for the graph G_r, we can construct in polynomial time a vertex cover of at most $|C_r| - \sum_{u \in V_{>3}}(deg(u) - 3)$ vertices for the graph G_1. Moreover, the size of a minimum vertex cover of the graph G_r is equal to the size of a minimum vertex cover* *of the graph* G_1 *plus* $\sum_{u \in V_{>3}}(deg(u) - 3)$.

Using the unfolding operations, we can prove

Lemma V.4 *The* P-VC-3 *problem has no parameterized algorithm of time* $2^{o(\sqrt{k})}n^{O(1)}$, *unless the* P-VC *problem has a parameterized algorithm of time* $2^{o(\sqrt{k})}n^{O(1)}$.

PROOF. Suppose the P-VC-3 problem has a parameterized algorithm A of time $2^{o(\sqrt{k})}n^{O(1)}$. We have the following algorithm A' shown in Fig 4 for the P-VC problem.

Algorithm A'

Input: A planar graph $G_1 = (V_1, E_1)$, $V_1 = V_{\leq 3} \cup V_{>3}$, and an integer $k > 0$.

Output: Output "Yes", if the size of the minimum vertex cover OPT_1 of G_1 satisfies $|OPT_1| \leq k$. Otherwise, output "No".

begin

1. Let $V_{>3}$ be the set of all vertices of degree larger than 3 in the graph G_1. Construct a planar graph G_2 by unfolding all vertices of degree larger than 3 in G_1.

2. Run the algorithm A on the graph G_2 with the parameter $k_2 = 1, 2, ..., |V_2|$. We get a minimum vertex cover OPT_2 for the graph G_2.

3. Construct a vertex cover OPT_1 for the graph G_1 from OPT_2 such that $|OPT_1| = |OPT_2| - \sum_{u \in V_{>3}}(deg(u) - 3)$.

4. If $|OPT_1| \leq k$, **Return** "Yes"; Otherwise, **Return** "No".

end

Fig. 4. Parameterized algorithm for PLANAR VERTEX COVER.

We prove the algorithm A' is correct. By Claim 2, OPT_1 is a vertex cover for the graph G_1 with $|OPT_2| - \sum_{u \in V_{>3}}(deg(u) - 3)$ vertices and OPT_1 is computable in time $n^{O(1)}$. Since OPT_2 is a minimum vertex cover for the graph G_2, by Claim 2 again, a minimum vertex cover for the graph G_1 contains $|OPT_2| - \sum_{u \in V_{>3}}(deg(u) - 3)$ vertices. In conclusion, OPT_1 is a minimum vertex cover for the graph G_1.

We analysis the running time of A' in the following.

For the graph $G_1 = (V_1, E_1)$, $V_1 = V_{\leq 3} \cup V_{>3}$, where $|V_1| = n$ and $|E_1| = m$, we can always assume $|OPT_1| \geq n/2$ by applying the NT-theorem [26]. That is, the parameter $k \geq n/2$. After applying the unfolding operation on each $v \in V_{>3}$, we get the new planar graph $G_2 = (V_2, E_2)$ with degree bounded by 3. The construction of G_2 can be done in polynomial time.

For a planar graph with n vertices and m edges, we have [32]:

$$m \leq 3n - 6. \tag{5.1}$$

By 5.1, for the graph G_1, the total degree of all its vertices satisfies:

$$\sum_{v \in V_1} deg(v) = 2m \leq 2(3n - 6) < 6n, \tag{5.2}$$

We have

$$\begin{aligned} |V_2| = |V_{\leq 3}| + \sum_{v \in V_{>3}} &((deg(v) - 3) + (deg(v) - 2)) \\ &< |V_{\leq 3}| + 2 \sum_{v \in V_{>3}} deg(v) \\ &\leq |V_1| + 2 \sum_{v \in V_1} deg(v) \\ &\leq n + 12n = 13n = O(n). \end{aligned}$$

Therefore, the calls to the algorithm A on the graph G_2 takes time $2^{o(\sqrt{|V_2|})}|V_2|^{O(1)} = 2^{o(\sqrt{n})}n^{O(1)} = 2^{o(\sqrt{k})}n^{O(1)}$. All the other steps of the algorithm A' takes polynomial time $n^{O(1)}$. Therefore the algorithm A' has running time $2^{o(\sqrt{k})}n^{O(1)}$. $\square$

Therefore, from Lemma V.3, Lemma V.4 and Theorem V.2, we have

Theorem V.5 *The* P-VC-3 *problem has no EPTAS of running time* $2^{o(\sqrt{1/\epsilon})}n^{O(1)}$*, where* $\epsilon > 0$ *is the given error bound, unless all SNP problems are solvable in subexponential time.*

Theorem V.5 implies the difficulty of improving the EPTAS algorithm for the P-VC-3 problem.

Baker [7] provided an EPTAS algorithm of time $2^{O(1/\epsilon)}p(n)$ for the P-VC problem. By applying that algorithm, we get an EPTAS algorithm of time $2^{O(1/\epsilon)}p(n)$ for the P-VC-3 problem. Since the P-VC-3 problem seems simpler, one might suspect that we could have a better EPTAS algorithm for it than that for the P-VC problem.

In the following we show that if we can improve the EPTAS algorithm for the P-VC-3 problem, then we can improve the EPTAS algorithm for the P-VC problem.

Theorem V.6 *If the* P-VC-3 *problem has an EPTAS of running time* $f(1/\epsilon)n^{O(1)}$*, then the* P-VC *problem has an EPTAS of running time* $f(13/\epsilon)n^{O(1)}$*, where* f *is a recursive function and* $\epsilon > 0$ *is the given error bound.*

PROOF. Given an EPTAS algorithm A of running time $f(1/\epsilon)n^{O(1)}$ for the P-VC-3 problem, we provide an EPTAS algorithm B of running time $f(13/\epsilon)n^{O(1)}$ for the P-VC problem. The description of algorithm B is given in Fig. 5.

We claim that the vertex set C_1 is the required vertex cover for the graph G_1.

By 5.1 and Claim 2, we have

$$\begin{aligned} |OPT_2| &= |OPT_1| + \sum_{u \in V_{>3}} (deg(u) - 3) \\ &\leq |OPT_1| + \sum_{u \in V_1} deg(u) \\ &\leq |OPT_1| + 6n \end{aligned}$$

Algorithm B

Input: A planar graph $G_1 = (V_1, E_1)$, and a constant $\epsilon > 0$.

Output: A vertex cover C_1 for G_1, such that $|C_1| \leq (1 + \epsilon) * |OPT_1|$.

begin

1. Let $V_{>3}$ be the set of all vertices of degree larger than 3 in the graph G_1. Unfold all vertices of degree larger than 3 in G_1, let the resulting graph be $G_2 = (V_2, E_2)$, whose degree is bounded by 3.

2. Run the algorithm A with $\epsilon' = \epsilon/13$ on the graph G_2. We get a vertex cover C_2 for the graph G_2.

3. From C_2 construct a vertex cover C_1 of at most $|C_2| - \sum_{u \in V_{>3}}(deg(u) - 3)$ vertices for the graph G_1.

4. **Return** C_1.

end

Fig. 5. EPTAS algorithm for PLANAR VERTEX COVER.

$$\leq |OPT_1| + 12|OPT_1|$$

$$\leq 13|OPT_1|.$$

Therefore,

$$|OPT_2| \leq 13|OPT_1|. \tag{5.3}$$

By Claim 2, we have

$$|OPT_1| = |OPT_2| - \sum_{u \in V_{>3}} (deg(u) - 3)$$

and

$$|C_1| \leq |C_2| - \sum_{u \in V_{>3}} (deg(u) - 3)$$

Therefore, we have

$$|C_2| - |C_1| \geq |OPT_2| - |OPT_1|$$

or equivalently

$$|C_2| - |OPT_2| \geq |C_1| - |OPT_1|$$

From this, we derive immediately

$$\begin{aligned}
& |C_1|/|OPT_1| - 1 \\
= \;& (|C_1| - |OPT_1|)/|OPT_1| \\
\leq \;& (|C_2| - |OPT_2|)/|OPT_1| \\
\leq \;& 13(|C_2| - |OPT_2|)/|OPT_2| \\
= \;& 13(|C_2|/|OPT_2| - 1) \\
\leq \;& 13 * (\epsilon/13) \\
= \;& \epsilon.
\end{aligned}$$

Here we have used the assumption that $C_2|/|OPT_2| \leq 1 + \epsilon' = 1 + \epsilon/13$, and the fact

$|OPT_2| \geq 13|OPT_1|$.

The call of the algorithm A on the graph G_2 takes time $f(1/\epsilon')n^{O(1)}$. All the other steps of the algorithm B take polynomial time $n^{O(1)}$. Therefore, the running time of the algorithm B is $f(13/\epsilon)n^{O(1)}$, and the approximation ratio for the algorithm B is $1+\epsilon$. $\square$

CHAPTER VI

CONCLUSIONS

A. Summary

In this thesis, we study the structures of parameterized problems with respect to their parameterized tractability and the relationship between parameterized complexity and approximability. The study has offered powerful techniques for deriving strong computational lower bounds for parameterized algorithms and approximation algorithms. We discussed the applications of these techniques.

In chapter II, we gave characterizations of two important approximation classes FPTAS and EPTAS. We proved that an NP optimization problem has a fully polynomial-time approximation scheme if and only if the problem is efficiently fixed-parameter tractable. By enforcing a constraint of planarity on the W-hierarchy studied in parameterized complexity theory, we obtained a class of NP optimization problems, the planar W-hierarchy, and proved that all problems in this class have efficient polynomial-time approximation schemes. Our new characterization of FPTAS has a number of advantages over the previous characterizations of this approximation class. Our characterization of EPTAS, which is significantly different from the PTAS characterization of Khanna and Motwani [57], is the first attempt to a systematic investigation of the structural properties of this new but important approximation class. Moreover, as a byproduct of our result, we answered an open problem posed by Downey and Fellows [37].

In Chapter III, based on our study of the structural properties of parameterized complexity theory, we introduced the concept of linear fpt-reductions, and used it to derive tight computational lower bounds for many well-known NP-hard problems, such as the INDEPENDENT SET, CLIQUE and DOMINATING SET problems. We also derived computational lower bound results for some Non NP-hard problems in the class LOGNP.

In Chapter IV, we extended our techniques developed in parameterized complexity to derive computational lower bounds for PTAS algorithms for NP-hard optimization problems, such as the DISTINGUISHING SUBSTRING SELECTION problem and the LONGEST COMMON SUBSEQUENCE problem. This seems to open a new direction for the study of computational lower bounds on the approximability of NP-hard optimization problems. We then discussed the inapproximability of the LOGNP problems. Our inapproximation result for V-C DIMENSION answered an open problem posed in literature.

In Chapter V, we derived computational lower bounds for EPTAS algorithms for some well-known NP-hard problems on planar graphs, such as PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and PLANAR DOMINATING SET. Since there is a gap between our lower bound results and the current upper bound results, in particular, we investigated the possibility of improving the upper bound of the EPTAS algorithm for the PLANAR VERTEX COVER problem. Our study showed that any asymptotic improvement on the EPTAS algorithms for the VERTEX COVER problem on planar graphs of degree bounded by 3 will result in an improvement on the EPTAS algorithms for the problem on general planar graphs.

B. Future Work

There seems to be intrinsic and interesting connections between the approximability and parameterized complexity of NP optimization problems. In Chapter II of this thesis we have studied the characterizations of the two approximation classes FPTAS and EPTAS using parameterized complexity theory. The relationship between the approximation class APX (the class of optimization problems that have constant-ratio polynomial time approximation algorithms) and the parameterized class FPT is worth exploring. For example, the problems in the class MAX SNP introduced by Papadimitriou and Yannakakis [65] and the class MIN $F^+ \pi_1$ introduced by Kolaitis and Thakur [59], are constant-ratio approximable, that is, in the class APX. In [12], Cai and Chen proved that all maximization problems in the class MAX SNP and all minimization problems in the class MIN $F^+ \pi_1$ are fixed-parameter tractable. We would like to define a set of optimization problems such that the problem is in APX if and only if the corresponding parameterized problem is in FPT. In [8], the author introduced the definition of covering problems, which include VERTEX COVER, K-SET COVER, HYPERGRAPH VERTEX COVER, FEEDBACK VERTEX SET on undirected graphes, and gave a unified approach for approximating these problems to constant ratios. We conjecture that if limited to covering problems, we can show that the class of APX is equal to the class of FPT. This research work might also throw light on the well-known problem in approximation area of getting an approximation ratio better than 2 for the VERTEX COVER problem.

Based on the study in chapter III of this thesis, we can introduce variants of fpt-reductions, such as linear fpt-reduction, simple fpt-reduction, and linear simple fpt-reduction to prove computational lower bounds for parameterized algorithms. We point out that the difference between these reductions in parameterized complexity and the ratio-preserving L-reduction in approximation, and the classical polynomial time reduction in NP-completeness theory is worth studying. The following are several simple observations. A polynomial time reduction from problem A to problem B can not guarantee an linear fpt-reduction since the parameters of the two problems may not be linearly related. Linear fpt-reductions are not sufficient to demonstrate a problem is NP-complete, for the reason that the reductions may be exponential in k [35]. The linear fpt-reductions are not L-reductions, as is discussed in chapter IV. We would like to further explore the relations between these reductions.

We are interested in the structural properties of parameterized complexity theory. In classical complexity, if the lower level of the polynomial hierarchy collapses, it would imply the collapse of the higher levels. That is, the polynomial hierarchy has what is called the "upward collapse" property. However, it is still open for the W-hierarchy in parameterized complexity whether $W[t] = \text{FPT}$ would imply $W[t+1] = \text{FPT}$. Based on our work in this thesis, we can see that such an upward collapse theorem is unlikely to hold for the W-hierarchy, as explained as follows. Suppose $W[t] = \text{FPT}$ implies $W[t+1] = \text{FPT}$. By Theorem III.10, if the $W[t+1]$-complete problem $\text{WCS}^*[t+1]$ is solvable in time $f_1(k)n^{o(k)}$ for a recursive function f_1, then $W[t] = \text{FPT}$, which by the assumed upward collapse theorem, would imply $W[t+1] = \text{FPT}$. In consequence, the problem $\text{WCS}^*[t+1]$ would be solvable in time $f_2(k)n^{O(1)}$. Thus, the upward collapse theorem would imply the following result:

> The problem $\text{WCS}^*[t+1]$ either can be solved in time $f_2(k)n^{O(1)}$ for a recursive function f_2, or cannot be solved in time $f_1(k)n^{o(k)}$ for any recursive function f_2.

Note that this result would be *unconditional*, i.e., not dependent of any complexity theory hypothesis. We feel that this would be a very strong result and if true, may require new and breakthrough techniques in complexity theory. For example, this would mean that if we could find a clique of size k in a graph of n vertices in time $n^{o(k)}$, then we would also be able to find the clique in time $f(k)n^c$ for a constant c. This invites further research work.

In future, we would like to explore the applications of our techniques for proving computational lower bounds for parameterized algorithms and approximation algorithms for other important problems. One example is the MOTIF FINDING problem, which has applications in finding conserved regions in molecular biology, as well as applications in coding theory [61]. A graph theoretical formulation of the MOTIF FINDING problem was proposed in [69]. It reduces the MOTIF FINDING problem to finding a maximum clique in a k-partite graph. According to the parameterized complexity theory, it has been proved in [76, 77] that this problem formulation is $W[1]$-complete with respect to the number of strings k as the parameter. We can derive computational lower bounds of the parameterized algorithms for this problem based on our work in the thesis. We are working on the parameterized complexity of the problem with respect to the maximum allowed Hamming distance d. The maximum allowed Hamming distance d is considered as the value of the objective function in designing a polynomial-time approximation scheme in [61]. If we can prove that the problem is $W[1]$-hard with respect to the parameter d, this would imply that the PTAS algorithm proposed in [61] could not be improved to an approximation algorithm of practical use. To resolve the parameterized complexity of this problem with respect to the parameter d will answer the open problem posed in [39, 46, 45]. Another interesting problem for further research, as we pointed out in Chapter V, is to close the gap between our lower bound results and the current upper bound results for the EPTAS algorithms for NP-hard problems on planar graphs.

REFERENCES

[1] 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.

[2] 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.

[3] J. Alber, H. Fernau, R. Niedermeier, "Parameterized complexity: exponential speed-up for planar graph problems," *J. Algorithms*, vol. 52, pp. 26-56, 2004.

[4] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *Journal of the ACM*, vol. 45, pp. 753-782, 1998.

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

[6] G. Ausiello, A. Marchetti-spaccamela, and M. Protasi, "Toward a unified approach for the classification of NP-complete optimization problems," *Theoretical Computer Science*, vol. 12, pp. 83-96, 1980.

[7] B.S. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *Journal of the ACM*, vol. 41, pp. 153-180, 1994.

[8] R. Bar-Yehuda, "One for the Price of Two: a Unified Approach for Approximating Covering Problems," *Algorithmica*, vol. 27, pp. 131-144, 2000.

[9] R. Beigel, "Finding maximum independent sets in sparse and general graphs," in *Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 856-857, 1999.

[10] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham, "Parameterized complexity analysis in computational biology," *Computer Applications in the Biosciences*, vol. 11, pp. 49-57, 1995.

[11] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham, "The parameterized complexity of sequence alignment and consensus," *Theor. Comput. Sci.*, vol. 147, pp. 31-54, 1995.

[12] L. Cai and J. Chen, "On fixed-parameter tractability and approximability of NP optimization problems," *Journal Of Computer and System Sciences*, vol. 54, pp. 465-474, 1997.

[13] L. Cai, J. Chen, R. Downey, and M. Fellows, "On the structure of parameterized problems in NP," *Information and Computation*, vol. 123, pp. 38-49, 1995.

[14] L. Cai, M. Fellows, D. Juedes, and F. Rosamond, "On effcient polynomial-time approximation schemes for problems on planar structures," *Manuscript of Unpublished Paper*, 2002.

[15] L. Cai and D. W. Juedes, "On the existence of sub-exponential time parameterized algorithms," *Journal of Computer and System Sciences*, vol. 67, pp. 789-807, 2003.

[16] L. Cai, D. W. Juedes, and I. Kanj, "The inapproximability of non-NP-hard optimization problems," *Theoretical Computer Science*, vol. 289, pp. 553-571, 2002.

[17] M. Cesati and L. Trevisan, "On the efficiency of polynomial time approximation schemes," *Information Processing Letters*, vol. 64, pp. 165-171, 1997.

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

[19] J. Chen, "Characterizing parallel hierarchies by reducibilities," *Information Processing Letters*, vol. 39, pp. 303-307, 1991.

[20] J. Chen, "Parameterized computation and complexity: a new approach dealing with NP-hardness," *Survey*, 2004.

[21] J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, "Tight lower bounds for parameterized NP-hard problems," in *Proc. of the 19th Annual IEEE Conference on Computational Complexity*, pp. 150-160, 2004.

[22] J. Chen, X. Huang, I. Kanj, and G. Xia, "Strong lower bounds on time complexity of PTAS for certain computational biology problems," *Manuscript of Unpublished Paper*, 2003.

[23] J. Chen, X. Huang, I. Kanj, and G. Xia, "Linear FPT reductions and computational lower bounds," in *Proc. of the 36th ACM Symposium on Theory of Computing*, pp. 212-221, 2004.

[24] J. Chen, X. Huang, I. Kanj, and G. Xia, "Polynomial time approximation schemes and parameterized complexity," *Lecture Notes in Computer Science*, vol. 3153, pp. 500-512, 2004.

[25] J. Chen, X. Huang, I. A. Kanj, and G. Xia, "Strong computational lower bounds via parameterized complexity," *Tech. Report*, Department of Computer Science,

Texas A&M University, 2004.

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

[27] J. Chen and A. Miranda, "A polynomial time approximation scheme for general multiprocessor job scheduling," *SIAM Journal on Computing*, vol. 31, pp. 1-17, 2001.

[28] Y. Chen and J. Flum, "Machine characterizations of the classes of the W-hierarchy," *Lecture Notes in Computer Science*, vol. 2803, pp. 114-127, 2003.

[29] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progression," *Journal of Symbolic Computation*, vol. 9, pp. 251-280, 1990.

[30] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang, "A PTAS for distinguishing (sub)string selection," *Lecture Notes in Computer Science*, vol. 2380, pp. 740-751, 2002.

[31] 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.

[32] R. Diestel, *Graph Theory*, New York: Springer, 2000.

[33] R. Downey, "Parameterized complexity for the skeptic," in *Proc. 18th IEEE Annual Conference on Computational Complexity*, pp. 132-153, 2003.

[34] R. 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.

[35] R. Downey, P. Evans, and M. Fellows, “Parameterized Learning Complexity,” in *Proc. 6th ACM Workshop on Computational Learning Theory*, pp. 51-57, 1993.

[36] R. Downey and M. Fellows, “Parameterized computational feasibility,” in *Proc. of the Second Cornell Workshop on Feasible Mathematics*, (Feasible Mathematics II, P. Clote and J. Remmel eds.), Birkhauser Boston, pp. 219-244, 1995.

[37] R. Downey and M. Fellows, *Parameterized Complexity*, New York: Springer-Verlag, 1999.

[38] M. Fellows, “Parameterized complexity: the main ideas and some research frontiers,” *Lecture Notes in Computer Science*, vol. 2223, pp. 291-307, 2001.

[39] M. Fellows, J. Gramm, and R. Niedermeier, “On the parameterized intractability of CLOSEST SUBSTRING and related problems,” *Lecture Notes in Computer Science*, vol. 2285, pp. 262-273, 2002.

[40] J. Flum and M. Grohe, “Describing parameterized complexity classes,” *Lecture Notes in Computer Science*, vol. 2285, pp. 359-371, 2002.

[41] F. V. Fomin and D. M. Thilikos, “Dominating sets in planar graphs: branch-width and exponential speed-up,” in *Proc. of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 168-177, 2003.

[42] F. V. Fomin and D. M. Thilikos, “A simple and fast approach for solving problems on planar graphs,” *Lecture Notes in Computer Science*, vol. 2996, pp. 56-67, 2004.

[43] M. Frick and M. Grohe, “Deciding first-order properties of locally tree-decomposable structures,” *Journal of the ACM*, vol. 48, pp. 1184-1206, 2001.

[44] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[45] J. Gramm, R. Niedermeier, and P. Rossmanith, "Fixed-parameter algorithms for CLOSEST STRING and related problems," *Algorithmica*, vol. 37, pp. 25-42, 2003.

[46] J. Gramm, J. Guo, and R. Niedermeier, "On exact and approximation algorithms for distinguishing substring selection," *Lecture Notes in Computer Science*, vol. 2751, pp. 195-209, 2003.

[47] M. Grohe, "Parameterized complexity for the database theorist," *SIGMOD Record*, vol. 31, pp. 86-96, 2002.

[48] J. Hastad, *Computational Limitations for Small-Depth Circuits*, The MIT Press, Cambridge, MA, 1986.

[49] M. T. Hallett, *An Integrated Complexity Analysi of Problems for Computational Biology*, Ph.D. Thesis, University of Victoria, 1996.

[50] D. S. Hochbaum, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[51] J. E. Hopcroft and R. E. Tarjan, "Efficient planarity testing," *Journal of the ACM*, vol. 21, pp. 549-568, 1974.

[52] O. H. Ibarra and C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," Journal of the ACM, vol. 22, pp. 463-468, 1975.

[53] R. Impagliazzo, R. Paturi, and F. Zane, "Which problems have strongly exponential complexity?" *J. Comput. Syst. Sci.*, vol. 63, pp. 512-530, 2001.

[54] T. Jian, "An $O(2^{0.304n})$ algorithm for solving maximum independent set problem," *IEEE Transactions on Computers*, vol. 35, pp. 847-851, 1986.

[55] T. Jiang and M. Li, "On the approximation of shortest common supersequence and longest common subsequences," *SIAM Journal on Computing*, vol. 24, pp. 1112-1139, 1995.

[56] I. Kanj and L. Perkovic, "Improved parameterized algorithms for planar dominating set," *Lecture Notes in Computer Science*, vol. 2420, pp. 399-410, 2002.

[57] S. Khanna and R. Motwani, "Towards a syntactic characterization of PTAS," in *Proc. 28th Annual ACM Symp. on Theory of Computing*, pp. 468-477, 1996.

[58] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, "On syntactic versus computational views of approximability," *SIAM Journal on Computing*, vol. 28, pp. 164-191, 1998.

[59] P. Kolaitis and M. Thakur, "Approximation Properties of NP Minimization Classes." *J. Comput. Syst. Sci.*, vol. 50, pp. 391-411, 1995.

[60] D. Maier, "The complexity of some problems on subsequences and supersequences," *Jounal of the ACM*, vol. 25, pp. 322-336, 1978.

[61] M. Li, B. Ma, and L. Wang, "On the closest string and substring problems," *Jounal of the ACM*, vol. 49, pp. 157-171, 2002.

[62] R. J. Lipton, R. E. Tarjan, "Applications of a planar separator theorem," *SIAM J. Comput.*, vol. 9, pp. 615-627, 1980.

[63] J. Nešetřil and S. Poljak, "On the complexity of the subgraph problem," *Commentationes Mathematicae Universitatis Carolinae*, vol. 26, pp. 415-419, 1985.

[64] R. Niedermeier and P. Rossmanith, "Upper bounds for vertex cover further improved," *Lecture Notes in Computer Science*, vol. 1563, pp. 561-570, 1999.

[65] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Journal Of Computer and System Sciences*, vol. 43, pp. 425-440, 1991.

[66] C. 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.

[67] C. Papadimitriou and M. Yannakakis, "On the complexity of database queries," *Journal of Computer and System Sciences*, vol. 58, pp. 407-427, 1999.

[68] A. Paz and S.Moran, "Non deterministic polynomial optimization problems and their approximations," *Theoretical Computer Science*, vol. 15, pp. 251-277, 1981.

[69] P. A. Pevzner and S.-H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," in *Proc. 8th International Conference on Intelligent Systems for Molecular Biology*, pp. 269-278, 2000.

[70] K. Pietrzak, "On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems," *Journal of Computer and System Sciences*, vol. 67, pp. 757-771, 2003.

[71] J. Robson, "Algorithms for maximum independent sets," *Journal of Algorithms*, vol. 7, pp. 425-440, 1986.

[72] J. Robson, "Finding a maximum independent set in time $O(2^{n/4})$?" LaBRI, Universite BordeauxI, 1251-01, 2001.

[73] C. Roth-Korostensky, *Algorithms for Building Multiple Sequence Alignments and Evolutionary Trees*, Ph.D. Thesis, No. 13550, ETH Zürich, 2000.

[74] S. Sahni, "Algorithms for scheduling independent tasks," *Journal of the ACM*, vol. 23, pp. 116-127, 1976.

[75] U. Stege, *Resolving Conflicts from Problems in Computational Biology*, Ph.D. Thesis, No. 13364, ETH Zürich, 2000.

[76] S.-H. Sze and J. Chen, "Finding specific motifs in DNA sequences via cliques in k-partite graphs," *Manuscript of Unpublished Paper*, 2003.

[77] S.-H. Sze, S. Lu, and J. Chen, "Integrating sample-driven and pattern-driven approaches in motif finding," in *Proc. 4th Workshop on Algorithms in Bioinformatics*, accepted, 2004.

[78] R. Tarjan and A. Trojanowski, "Finding a maximum independent set," *SIAM Journal on Computing*, vol. 6, pp. 537-546, 1977.

[79] G. Woeginger, "When does a dynamic programming formulation guarantee the existence of an FPTAS?" in *Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 820-829, 2001.

VITA

Name: Xiuzhen Huang

Address: Computer Science Department, P.O. Box 9, State University, AR 72467

Email: xzhuang@csm.astate.edu

Education: M.S. in computer science, Shandong University, China, July 1999;

B.S. in computer science, Shandong University, China, July 1996.

The typist for this thesis was Xiuzhen Huang.