Jan. 2005, Vol.20, No.1, pp.18–37 J. Comput. Sci. & Technol.

Parameterized Computation and Complexity: A New Approach Dealing with NP-Hardness

Jian-Er Chen

College of Information Science and Engineering, Central South University, Changsha 410083, P.R. China

E-mail: jianer@mail.csu.edu.cn

Received May 21, 2004; revised October 1, 2004.

Abstract The theory of parameterized computation and complexity is a recently developed subarea in theoretical computer science. The theory is aimed at practically solving a large number of computational problems that are theoretically intractable. The theory is based on the observation that many intractable computational problems in practice are associated with a parameter that varies within a small or moderate range. Therefore, by taking the advantages of the small parameters, many theoretically intractable problems can be solved effectively and practically. On the other hand, the theory of parameterized computation and complexity has also offered powerful techniques that enable us to derive strong computational lower bounds for many computational problems, thus explaining why certain theoretically tractable problems cannot be solved effectively and practically. The theory of parameterized computation and complexity has found wide applications in areas such as database systems, programming languages, networks, VLSI design, parallel and distributed computing, computational biology, and robotics.

This survey gives an overview on the fundamentals, algorithms, techniques, and applications developed in the research of parameterized computation and complexity. We will also report the most recent advances and excitements, and discuss further research directions in the area.

Keywords algorithm, computational complexity, NP-completeness, parameterized computation, approximation algorithm

1 Introduction

One of the greatest achievements in theoretical computer science is the development of NP-completeness theory[1]. NP-completeness theory provides a solid and convincing foundation for the study of intractable computational problems. However, the theory does not obviate the need for solving these hard problems because of their practical importance. Many approaches have been proposed, including polynomial time approximation algorithms[2], randomized algorithms[3], and heuristic algorithms[4]. None of these approaches has satisfied all needs requested from industry and applications: polynomial time approximation algorithms can only provide approximation solutions while certain applications may require precise solutions; the success of a randomized algorithm on a problem in general heavily depends on the probabilistic distribution of the problem instances; and heuristic algorithms in general do not have formal performance guarantees.

The theory of parameterized computation and complexity is a more recently developed new approach dealing with hard computational problems arising from industry and applications. The theory is based on the observation that many intractable computational problems in practice are associated with a parameter that varies within a small or moderate range. Therefore, by taking the advantages of the small parameters, many theoretically intractable problems can be solved effectively and practically. The following are some examples.

• The VERTEX COVER problem (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[1]. 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[5,6], 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 $O(1.286^k + kn)$[30]. This algorithm has been implemented and is quite practical[8].

• The ML TYPE-CHECKING problem (given an ML program P, determine whether type declarations in the program are consistent). Here the parameter k is the maximum nesting depth of the type declarations[9]. The problem is EXPTIME-complete[10] (thus is even harder than NP-complete problems). Normally, the number k of depth of nested declarations is not larger than 10. Algorithms of running time $O(2^k n)$ for this problem have been developed[9], which is clearly practical for solving the problem in most applications.

• The FAULT COVERAGE IN MEMORY ARRAY problem (given a reconfigurable rectangle memory array with defective elements, determine whether the array structure can be repaired by replacing k_1 rows and k_2 columns). Here the parameter is $k = k_1 + k_2$. The problem is

*Survey

This research is supported in part by the National Natural Science Foundation of China under Grants No.60373083 and No.60433020, and by the Changjiang Scholar Reward Project of Ministry of Education, P.R. China.

NP-complete[11]. A typical value for the parameter k is bounded by 40. The problem has been extensively studied in the last two decades, by both computer theoreticians and computer engineers. A recently developed algorithm for the problem[11] has its running time bounded by $O(1.26^k n)$, which is again quite practical.

Therefore, one research direction in parameterized computation and complexity theory is to fully take the advantages of small parameter values and develop practically effective algorithms for computational problems arising in applications which are theoretically intractable. We will discuss designing techniques and algorithmic results in parameterized computation in Section 3.

The rich positive toolkit of novel techniques for developing efficient parameterized algorithms is accompanied in the theory by a corresponding negative toolkit that supports a theory of parameterized intractability. This theory is based on the extensive research and experience in computational practice showing that certain parameterized problems remain intractable even if the parameter values are kept small. Consider the following problems.

• The INDEPENDENT SET problem (given a graph G and an integer k, determine whether G contains an independent set I of k vertices, i.e., a subset I of k vertices in G such that no two vertices in I are adjacent). Here the parameter is the integer k. The problem has applications in many areas such as network optimization and computational biology. The best algorithm solving this problem runs in time $O(n^{0.792k})$[12], based on currently the best algorithm for matrix multiplication[13].

• The DATABASE QUERY EVALUATION problem (given a query to a relational database, determine whether the query has the value *true*). Here the parameter is the size k of the query. The query size k is in general much smaller than the database size n. The best known algorithm for this important problem in database systems runs in time $O(n^k)$[14], which is not practical even for very small parameter values.

Note that an algorithm of running time $O(n^k)$ for solving the above problems is straightforward. For example, to solve the INDEPENDENT SET problem, we can simply enumerate all subsets of k vertices in the given graph of n vertices, and check if any of these subsets contains no adjacent vertices. The running time of this trivial algorithm is proportional to the number of subsets of k vertices in the whole set of n vertices, and is of the order of $\binom{n}{k}\binom{k}{2} = O(n^k)$. On the other hand, researchers have struggled for many years trying to develop "smarter" algorithms that avoid this exhaustive enumeration process. All these efforts have unfortunately failed. Very recent research[15] has shown strong evidences that for certain problems such as INDEPENDENT SET, neither polynomial time pre-processing nor thorough local optimization can help avoiding the exhaustive enumeration. Therefore, these problems seems to require computational time of order of $n^{\Theta(k)}$.

Therefore, though both being NP-complete, VERTEX COVER and INDEPENDENT SET seem to be significantly different in terms of their computational complexity on small parameter values. The $O(1.286^k + kn)$ time algorithm for VERTEX COVER makes the problem practically feasible for parameter values not larger than 60, while for INDEPENDENT SET, even for parameter values as small as 6, the $O(n^{0.792k})$ time algorithm is already not practical: for $n = 1,000$ and $k = 60$, $1.286^k + kn$ is less than 4×10^6, while for $n = 1,000$ and $k = 6$, $n^{0.792k}$ is larger than 18×10^{13}.

To distinguish these two different kinds of parameterized problems, a formal framework has been established in the theory of fixed-parameter tractability. The framework divides the parameterized problems into two fundamental classes: the class of FIXED-PARAMETER TRACTABLE problems (i.e., FPT problems), such as VERTEX COVER, and the class of *fixed-parameter intractable* problems (i.e., $W[1]$-hard problems), such as INDEPENDENT SET. This classification has been successfully used to establish computational lower bounds for a large number of problems of practical importance. For example, In a special issue of *Journal of Computer and System Sciences* (Volume 58, 1999), Papadimitriou and Yannakakis[14] used this framework and proved that the DATABASE QUERY EVALUATION problem is $W[1]$-hard. The preliminary version of this result won the Best Paper Award in the *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* This implies that any algorithm solving the problem will be impractical even for small query size. The Guest Editor of the special issue pointed out: "*parametric complexity theory is a productive framework for studying the complexity of query languages.*"

We should point out that the theory of fixed-parameter tractability is not a simple refinement of the concept of NP-completeness. As we have seen, there are fixed-parameter tractable problems, such as the ML TYPE-CHECKING problem, which are harder than NP-complete problems. On the other hand, there are also fixed-parameter intractable problems that seem easier than NP-complete problems. The following are two examples.

• V-C DIMENSION: given a collection $\mathcal{C}$ of subsets in a universal set U, and an integer k, decide whether there is a subset S of k elements in U such that for every subset T of S there is a subset $C_T \in \mathcal{C}$ such that $S \cap C_T = T$. Here the parameter is k.

The problem has important applications in computational learning theory[16]. It can be shown that the size of the subset S is always bounded by $\log |\mathcal{C}|$ (see [17]). Therefore, the problem can be solved in time $O(n^{O(\log |\mathcal{C}|)})$ by enumerating all subsets of size bounded by $\log |\mathcal{C}|$. In consequence, the problem is unlikely to be NP-hard. On the other hand, the problem is $W[1]$-hard (see [18]). Thus, even the problem seems easier than NP-complete problems, according to the theory of fixed-parameter tractability, it has no efficient algorithms even

for small parameter values.

• MOTIF FINDING: given k sample DNA sequences, find an (l, d)-motif (i.e., find a pattern of length l that matches a subsequence in each given sequence with d mismatches allowed). Here k is the parameter.

This is an important problem in computational biology. A typical value for the parameter k is 20. Recently, it has been proposed to solve the problem via solving the problem of finding a clique of no more than 20 vertices in a large graph G of n vertices (n is in general of the order of at least $10,000$)[19]. The problem is solvable in time $O(n^{20})$ by enumerating all subsets of no more than 20 vertices in the graph. Thus, the problem is polynomial time solvable. On the other hand, the problem is $W[1]$-hard[20]. This explains why this polynomial time solvable problem has no practically efficient algorithms.

Therefore, the theory of fixed-parameter tractability seems to well supplement the theory of NP-completeness. In particular, the theory approaches from a more practical angle, studies the techniques that may be effectively used in practice to solve certain theoretically intractable problems, and identifies problems that are theoretically tractable but have no practically effective algorithms.

We recommend the book by Downey and Fellows[18] to interested readers for a more systematic treatment of the theory of fixed-parameter tractability. The recently published special issue in *Journal of Computer and System Sciences* (Volume 67, No.6, 2003, Guest Editors: Chen J. and Fellows M.) provides updates and new developments in the area.

2 Fundamentals

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[1] to reduce an optimization problem to a decision problem by introducing a parameter.

Definition 1. *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 non-negative 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[18]. Our discussion will be based on the above simplified definition in which the parameter is a nonnegative integer, as it 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 2. *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 $1 \leqslant i \leqslant n$. 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.

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 3. *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 2.1. *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[18].

Definition 4. *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 $t \geqslant 1$. Moreover, observe that the WCNF-2SAT$^-$ problem is a subproblem of the WCS$[2]$ problem (thus, WCNF-2SAT$^-$ is trivially fpt-reducible to WCS$[2]$) 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* $\bigcup_{t\geqslant 0} W[t]$, 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 5. *Let $t \geqslant 1$ 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 2.2. *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 2.3. *For $t \geqslant 1$, if a $W[t]$-hard problem is fixed-parameter tractable, then $FPT = W[t]$.*

Since it is commonly believed that for all $t \geqslant 1$, $FPT \neq W[t]$, 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 2.4. *Let $t \geqslant 1$ 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 $t \geqslant 2$, it can be shown that the problem WCS$[t]$ is fpt-reducible to the problem WCS$^*[t]$[18]. The problem WCS$^*[t]$ is obviously in the class $W[t]$. Therefore, for each $t \geqslant 2$, we get the second $W[t]$-complete problem WCS$^*[t]$.

Using Theorem 2.4, 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[18]. 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 [21–23] for detailed discussions.

3 How Easy Can FPT Be: Algorithmic Techniques

We first describe how fixed-parameter tractability helps solving difficult computational problems. Our principle is "simpler computation and deeper theory"[24]. Therefore, we are not simply looking for purely theoretical improvements over parameterized algorithms based on the traditional asymptotical analysis. We also take the practicality of the algorithms into consideration. In particular, we insist that for practical effectiveness, our algorithms must be structurally simple, avoiding as much as possible lengthy case-by-case enumerations of tedious local structures. The improvements on the algorithms should be based on deep theoretical study of the intrinsic properties of the concerned problems. The techniques described in this section have proven effective in the development of efficient parameterized algorithms. Many of these developed algorithms have been implemented, and turn out to be quite practical in various applications.

3.1 Kernelization

Kernelization has been a well-known and effective technique used in the development of efficient parameterized algorithms. However, only very recently, people realized that kernelizability of a parameterized problem is equivalent to the fixed-parameter tractability of the problem. We first present a formal proof for this fact.

Definition 6. *A parameterized problem Q is kernelizable if there exist a polynomial time algorithm K (the "kernelization algorithm") and a recursive function g such that on any instance (x,k) of Q, the algorithm K produces an instance (x',k') of Q (the "kernel"), such that $|x'| \leqslant g(k)$ and $k' \leqslant k$, and that (x,k) is a yes-instance of Q if and only if (x',k') is a yes-instance of Q.*

The definition for kernelization given above is slightly stronger than those given in [25] and [26], in the sense that it requires that the reduction be strictly polynomial-time bounded. Therefore, the following theorem is also slightly stronger than the one given in [26]. Recall that a decision problem (or more precisely, a language) is *decidable* if it can be solved by an algorithm.

Theorem 3.1. *Let Q be a decidable parameterized problem. Then Q is fixed-parameter tractable if and only if Q is kernelizable.*

Proof. Since Q is decidable, there is an algorithm that solves Q. Let the running time of the algorithm be bounded by $h(|x|,k)$, where h can be any function. Without loss of generality, we assume that the function h is nondecreasing and unbounded. [①]

Suppose that Q is kernelizable. Then there exist a kernelization algorithm K and a function g as given in its definition. We solve the problem Q as follows. For a given instance (x,k) of Q, we call the kernelization algorithm K to construct the instance (x',k'), which takes time $O(|x|^c)$, where c is a constant. Then we solve the instance (x',k') in time $h(|x'|,k')$. Since $|x'| \leqslant g(k)$ and $k' \leqslant k$, we have $h(|x'|,k') \leqslant h(g(k),k) = h'(k)$, where $h'(k)$ is a function of k. Therefore, the problem Q can be solved in time $O(h'(k)+|x|^c)$, and hence is fixed-parameter tractable.

Conversely, suppose that Q is fixed-parameter tractable and can be solved by a parameterized algorithm M of running time bounded by $f(k)|x|^c$, where f is a nondecreasing and unbounded function and c is a constant. Fixed a yes-instance (x_Y,k_Y) of Q such that k_Y is the minimum over all yes-instances of Q, and a no-instance (x_N,k_N) of Q such that k_N is the minimum over all no-instances of Q. Consider the following algorithm K that on an input (x,k) produces an instance (x',k') of Q as follows. If $f(k) > |x|$, then $(x',k') = (x,k)$. In case $f(k) \leqslant |x|$, K first calls the algorithm M to determine if (x,k) is a yes-instance of Q. If (x,k) is a yes-instance of Q then K outputs (x_Y,k_Y), and if (x,k) is a no-instance of Q then K outputs (x_N,k_N). It is clear that the algorithm K outputs a yes-instance of Q if and only if the input (x,k) to K is a yes-instance of Q. Moreover, in case $f(k) \leqslant |x|$, the running time of M is bounded by $f(k)|x|^c \leqslant |x|^{c+1}$. Therefore, K is a polynomial time algorithm. Finally, in case $f(k) > |x|$, the output $(x',k') = (x,k)$ of K satisfies $|x'| \leqslant f(k)$ and $k' \leqslant k$. On the other hand, in case $f(k) \leqslant |x|$, we have either $(x',k') = (x_Y,k_Y)$ or (x_N,k_N). By the definitions of k_Y and k_N, we must have $k' \leqslant k$, and $|x'| \leqslant c_1$, where $c_1 = \max\{|x_Y|,|x_N|\}$ is a constant. In summary, on any input (x,k), the output (x',k') of K satisfies $k' \leqslant k$ and $|x'| \leqslant g(k)$, where $g(k) = f(k)+c_1$. This proves that K is a kernelization algorithm for Q, and hence Q is kernelizable. □

The first part of the proof for Theorem 3.1 has actually described a procedure for developing parameterized algorithms for a kernelizable problem. In particular, the smaller the size of the kernel (x',k'), the more efficient the resulting parameterized algorithm. Therefore, it is of both practical and theoretical interests to study how small the problem kernel can be for a fixed-parameter tractable problem.

We illustrate the technique of kernelization by an example. Recall that the VERTEX COVER problem is for a given pair (G,k), where G is a graph and k is a non-negative integer, to determine if G has a vertex cover C of k vertices (i.e., a set C of k vertices in G such that every edge in G has at least one end in C).

Suppose that the graph G has n vertices $v_1,\ldots,v_n$, we construct a new bipartite graph G' as follows. The graph G' has $2n$ vertices that are bi-partitioned into two sets $\{w_1,\ldots,w_n\}$ and $\{w'_1,\ldots,w'_n\}$. There is an edge $[w_i,w'_j]$ in the bipartite graph G' if and only if $[v_i,v_j]$ is an edge in G. Construct any minimum vertex cover C' of the bipartite graph G' (a minimum vertex cover of a bipartite graph G' of $2n$ vertices can be constructed in time $O(n^{2.5})$ via a maximum matching of the graph G', see [27, 28] for detailed discussions). Now divide the vertices in the original graph G into three disjoint subsets: set I_0 consists of all the vertices v_i in G such that neither w_i nor w'_i is in C'; set C_0 consists of all the vertices v_i in G such that both w_i and w'_i are in C'; and set V_0 consists of all the vertices v_i in G such that exactly one of w_i and w'_i is in C'.

Lemma 3.2[29]**.** *Let $G(V_0)$ be the subgraph of G induced by the vertex set V_0. Then every minimum vertex cover of $G(V_0)$ plus the set C_0 forms a minimum vertex cover for G. Moreover, a minimum vertex cover of $G(V_0)$ contains at least $|V_0|/2$ vertices.*

We explain how Lemma 3.2 is used to kernelize the VERTEX COVER problem. Given an instance (G,k) for

[①] Otherwise, we replace h by a larger function that is nondecreasing and unbounded. For the same reason, we will assume in this paper that all complexity functions in our discussion have "nice" properties such as that the domain and range of the functions are nonnegative integers, and that the values of the functions and their inverses can be computed in polynomial time.

vertex cover, apply Lemma 3.2 to the graph G to construct in polynomial time the two subsets C_0 and V_0. By Lemma 3.2, the graph G has a vertex cover of size k if and only if the induced subgraph $G(V_0)$ has a vertex cover of size $k - |C_0|$. Since the minimum vertex cover of the graph $G(V_0)$ consists of at least $|V_0|/2$ vertices, a necessary condition for the graph $G(V_0)$ to have a vertex cover of size $k - |C_0|$ is that the number $|V_0|$ of vertices of the graph $G(V_0)$ is bounded by $2k - 2|C_0|$. Let $G' = G(V_0)$ and $k' = k - |C_0|$, then we have constructed an instance (G', k') for VERTEX COVER, where G' has at most $2k' \leqslant 2k$ vertices and $k' \leqslant k$, such that the graph G' has a vertex cover of k' vertices if and only if the graph G has a vertex cover of k vertices. This gives a kernelization for the VERTEX COVER problem, formally stated as follows.

Theorem 3.3[30]. *There is a kernelization algorithm of running time $O(n^{2.5})$ for the* VERTEX COVER *problem that for a given instance (G, k) of* VERTEX COVER, *constructs another instance (G', k'), where the graph G' contains at most $2k'$ vertices and $k' \leqslant k$, such that the graph G has a vertex cover of k vertices if and only if the graph G' has a vertex cover of k' vertices.*

There are two directions to improve Theorem 3.3: one is to further reduce the kernel size, and the other is to improve the running time of the kernelization algorithm. Further reducing the kernel size seems difficult since it would imply an improvement over the currently best approximation algorithms for the VERTEX COVER problem, and hence resolving a very well-known open problem in the research of approximation algorithms[30]. On the other hand, the complexity of the kernelization algorithm in Theorem 3.3 can be improved to $O(kn + k^3)$ if another preprocessing is used before we apply Lemma 3.2[30].

The investigation of kernelization of parameterized problems has induced and motivated a number of new research directions in algorithm design and graph theory. A new technique, called the *crown reduction*[31], has been discovered in kernelizing parameterized problems, which suggests a simpler procedure to construct a structure similar to the one given in Lemma 3.2, as follows.

A *crown* is a pair (I_0, H_0) of disjoint subsets of vertices in a graph G satisfying the following conditions: (1) I_0 is an independent set; (2) H_0 consists of all the vertices adjacent to some vertices in I_0; and (3) there is a matching M of $|H_0|$ edges in G such that each edge in M has an end in I_0 and an end in H_0. It is easy to see that there is a minimum vertex cover of G that contains all vertices in H_0 but excludes all vertices in I_0. Therefore, we can simply work on searching for a vertex cover of size $k' = k - |H_0|$ in the subgraph $G' = G - I_0 - H_0$. Repeatedly applying this technique, it can be shown that a kernel of size bounded by $3k$ can be constructed for the VERTEX COVER problem, which seems a very different approach from Lemma 3.2 based on Nemhauser-Trotter's theorem[29]. Moreover, the crown reduction technique can be applied effectively to other parameterized problems such as the SAVE-k-COLORS problem[32].

Another interesting example for kernelization of parameterized problems is the study on PLANAR DOMINATING SET (i.e., determine for a given pair (G, k), where G is a planar graph and k is an integer, whether G as a dominating set of k vertices). Parameterized algorithms on PLANAR DOMINATING SET have drawn much attention in the study of parameterized computation and complexity (see Subsection 3.3). The study has induced a highly nontrivial and subtle kernelization algorithm, with a kernel size bounded by $335k$, for the problem[33].

3.2 Bounded Search-Trees

The branch-and-search process based on bounded search-trees has been a very popular technique in the development of exact algorithms and heuristic algorithms for difficult computational problems[4,34–36]. The technique has also been a major tool in developing efficient parameterized algorithms for fixed-parameter tractable problems.

A typical branch-and-search parameterized algorithm M solving a fixed-parameter tractable problem Q works in the following recursive manner: to determine if a nontrivial instance (x', k') is a yes-instance of Q, the algorithm M constructs instances $(x_1, k_1), \ldots, (x_s, k_s)$ of smaller parameter values, solves these instances recursively, and finally derives a solution to (x', k') based on the solutions to the instances of smaller parameter values. This process can be depicted as a bounded search-tree $\mathcal{T}$, as follows. The execution of the algorithm M on the instance (x', k') corresponds to a node labelled (x', k') in $\mathcal{T}$, with s children labelled (x_1, k_1), ..., (x_s, k_s), respectively, corresponding to the s recursive executions of M on these instances induced in the execution of M on (x', k'). The root of the search-tree $\mathcal{T}$ is labelled by the original input (x, k) to the algorithm M, and each leaf of $\mathcal{T}$ is labelled by an instance which is simple enough to be solved directly.

Therefore, the branch-and-search algorithm M is a traversing process in the search-tree $\mathcal{T}$, typically in the Depth-First-Search order, and its time complexity is proportional to the size of the search-tree $\mathcal{T}$. Since the size of $\mathcal{T}$ is bounded by twice of the number of leaves in $\mathcal{T}$ (we suppose that each non-leaf node in $\mathcal{T}$ has at least two children), we can use the number of leaves in $\mathcal{T}$ to evaluate the computational complexity of the branch-and-search algorithm. On the node τ labelled (x', k') in $\mathcal{T}$ with s children labelled $(x_1, k_1), \ldots, (x_s, k_s)$, if we let $T(k')$ denote the number of leaves in the subtree rooted at τ, then we have the following recurrence relation:

$$T(k') = T(k_1) + \cdots + T(k_s). \tag{1}$$

The recurrence relation (1) is solved as follows (see [35] for more detailed discussions). Suppose without loss of generality that $k' > k_1 \geqslant k_2 \geqslant \cdots \geqslant k_s$. Let $T(x) = \alpha^x$, where α is a constant to be determined. Then the recurrence relation (1) becomes the following polynomial

equation:

$$P(\alpha) = \alpha^{k'-k_s} - \alpha^{k_1-k_s} - \cdots - \alpha^{k_{s-1}-k_s} - 1 = 0. \quad (2)$$

Note that each non-leaf node in $\mathcal{T}$ corresponds to a polynomial equation of the form (2). It can be proved that this polynomial equation has exactly one positive root α' and that for all values $\alpha > \alpha'$, we have $P(\alpha) < 0$[30]. Therefore, if we let α_0 be the largest positive real number such that α_0 is the root of a polynomial equation corresponding to a non-leaf node in the bounded search-tree $\mathcal{T}$, then it is easy to verify using induction that $T(k') \leqslant \alpha_0^{k'}$ for any non-leaf node labelled (x', k') in the search-tree $\mathcal{T}$. In particular, the inequality holds for the root of the search-tree $\mathcal{T}$. Hence, α_0^k serves as an effective upper bound on the computational complexity of the branch-and-search algorithm M on the instance (x, k).

We again use VERTEX COVER as a concrete example to illustrate this technique. Let (G, k) be an instance of VERTEX COVER. First note that if all vertices in the graph G have degree bounded by 2, then each connected component of G is either a simple cycle or a simple chain, and hence it can be decided in linear time whether (G, k) is a yes-instance for VERTEX COVER. Therefore, we can assume without loss of generality that the graph G has a vertex of degree larger than 2. To determine if the graph G has a vertex cover C of k vertices, we pick any vertex v of degree $d \geqslant 3$ in G and "branch at v" by trying the following two possibilities: either (1) v is in the vertex cover C thus there should be a vertex cover of $k_1 = k - 1$ vertices in the graph $G_1 = G - \{v\}$; or (2) v is not in the vertex cover C, then all the d neighbors $u_1, \ldots, u_d$ of v must be in the vertex cover C (in order to cover the edges $[v, u_1]$, ..., $[v, u_d]$) and there should be a vertex cover of $k_2 = k - d$ vertices in the graph $G - \{v, u_1, \ldots, u_d\}$. Therefore, by recursively working on the instances (G_1, k_1) and (G_2, k_2), we can determine if (G, k) is a yes-instance for VERTEX COVER. The recurrence relation corresponding to this branching process is $T_d(k) = T_d(k-1) + T_d(k-d)$, and the associated polynomial equation is $P_d(\alpha) = \alpha^d - \alpha^{d-1} - 1$. Solving the polynomial equation $P_d(\alpha) = 0$, we get a root $\alpha_d > 0$. In particular, we have $\alpha_3 = 1.465\ldots$, $\alpha_4 = 1.380\cdots$, $\alpha_5 = 1.324\cdots$, $\alpha_6 = 1.285\cdots$, $\alpha_7 = 1.255\cdots$, and $\alpha_d < 1.255$ for $d > 7$.

This hints immediately a branch-and-search algorithm for VERTEX COVER: for a given instance (G, k), recursively branch at a vertex of degree larger than 2, until there is no vertex of degree larger than 2 in the graph, which now can be solved in linear time. Since the largest real number that can be a root of a polynomial equation induced by this branch-and-search algorithm is $\alpha_3 = 1.465\cdots$, we conclude that the size of the bounded search-tree for the algorithm is bounded by 1.47^k. This, plus a general technique in analyzing parameterized algorithms[37], derives a parameterized algorithm of running time bounded by $O(1.47^k + kn)$ for the VERTEX COVER problem.

The above discussion shows that the branch-and-search algorithm for VERTEX COVER slows down when it branches at low degree vertices. To further speed up the algorithm, considerable efforts have been made to deal with low degree graphs[25,30,38–40]. Intuitively, low degree graphs have a simpler structure so that we may take more effective manipulations. In the following, we illustrate one of such techniques, and present a more efficient parameterized algorithm for VERTEX COVER on graphs of degree bounded by 3.

Suppose v is a degree-2 vertex in a graph G such that the two neighbors u and w of v are not adjacent to each other. We construct a new graph G' as follows: remove the vertices v, u, and w and introduce a new vertex v_0 that is adjacent to all neighbors of the vertices u and w in G (of course except the vertex v). We say that the graph G' is obtained from the graph G by *folding the degree-2 vertex v*. See Fig.1 for an illustration for this operation.

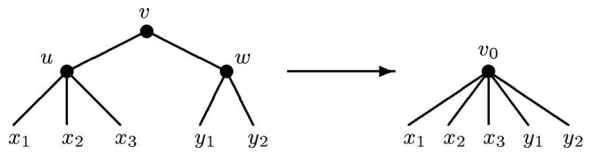

Fig.1. Vertex folding.

Lemma 3.4[30]. *Let G' be a graph obtained by folding a degree-2 vertex v in a graph G, where the two neighbors of v are not adjacent to each other. Then the graph G has a vertex cover C of k vertices if and only if the graph G' has a vertex cover C' of $k-1$ vertices. Moreover, the vertex cover C for G can be constructed from the vertex cover C' for G' in constant time.*

Therefore, we can work on the instance $(G', k-1)$ instead of (G, k), and the vertex folding operation reduces the parameter value k by 1 directly without any branching. There are another two possible cases when the graph G has a vertex of degree less than 3: 1) the graph G has a degree-1 vertex v. Then we can simply include the neighbor of v in the vertex cover; and 2) the graph G has a degree-2 vertex v whose two neighbors u and w are adjacent to each other. In this case, since every vertex cover of G must contain at least two vertices in the triangle (v, u, w), we can simply include the vertices u and w in the vertex cover. In both cases, we can do at least as well as vertex folding. Therefore, whenever the graph G has a vertex of degree less than 3, we can always assume that the vertex folding operation (or a better operation) is applicable.

This suggests a branch-and-search parameterized algorithm for VERTEX COVER on graphs of degree bounded by 3, as given in Fig.2. By a simple preprocessing, we can assume that the original input graph G to the algorithm **vc-degree3** contains a vertex of degree less than 3 (See [30] for a detailed explanation). Note that this condition is automatically satisfied for each intermediate recursive execution of the algorithm, since any proper subgraph of

a graph of degree bounded by 3 must contain a vertex of degree less than 3.

vc-degree3
1. $C = \emptyset$; {initialize the vertex cover C}
2. **while** $|C| < k$ **and** G is not empty **do**
2.1. pick a degree-2 vertex v in G and fold v;
2.2. **if** the new vertex v_0 has a degree larger than 2 **then** branch at v_0.

Fig.2. Algorithm for VERTEX COVER on graphs of degree bounded by 3.

We consider the running time of the algorithm **vc-degree3**. According to the given condition, before branching at a vertex, we can apply the vertex folding operation at least once, which reduces the parameter value by 1. Moreover, we always branch at a vertex of degree at least 3, which reduces the parameter value by 1 on one side and by at least 3 on the other side. Combining this with the vertex folding before the branching, we can regard this process as a branching which reduces the parameter value by 2 on one side and by at least 4 on the other side. Therefore, the branching can do at least as well as the following recurrence relation:

$$T(k) = T(k-2) + T(k-4).$$

The polynomial equation $x^4 - x^2 - 1 = 0$ corresponding to this recurrence relation has a root $\alpha = 1.272\cdots$. This, combined with other techniques[30], gives an $O(1.273^k + kn)$ time algorithm for the VERTEX COVER problem on graphs of degree bounded by 3.

This gives an immediate improvement on the algorithm for VERTEX COVER on general graphs: for a given general graph G, we can first branch at vertices of degree larger than 3 until the graph contains no such vertices, then we apply the algorithm **vc-degree3**. In this new algorithm, the worst recurrence relation is $T(k) = T(k-1) + T(k-4)$, which has a solution $T(k) \leqslant 1.381^k$, implying an algorithm of running time $O(1.39^k + kn)$ for VERTEX COVER on general graphs.

Other design and analysis techniques, by more carefully dealing with graphs of degree bounded by 4 and 5, can be applied to further improve the algorithm[30,38–40]. After many rounds of improvement, currently the best parameterized algorithm for VERTEX COVER[30] has its running time bounded by $O(1.286^k + kn)$. This algorithm has found important applications in computational biology and computational biochemistry[5,6], and has been implemented by a number of research and development groups in the world. The implementations show that the algorithm can effectively solve the VERTEX COVER problems for their applications (see, for example, [8]).

The advantage of branch-and-search algorithms for fixed-parameter tractable problems is that the parameter value k is in general small from their applications. Therefore, a parameterized algorithm of running time bounded by $O(c^k)$ plus a polynomial of n, where c is a small constant, can be very efficient in practical applications, even for a moderate value of k. For example, the $O(1.286^k + kn)$ time algorithm for the VERTEX COVER problem seems to be practical for parameter values up to $k = 400$ (see discussions in [8]).

3.3 Graph Separators

Graph separators have been used effectively in designing efficient algorithms, in particular, in the development of approximation algorithms for NP-hard optimization problems[41,42]. In this subsection, we show how the technique is used in the development of parameterized algorithms for certain fixed-parameter tractable problems. To be concrete, we take the PLANAR DOMINATING SET problem as an example, which is defined as follows: for a given pair (G, k), where G is a planar graph and k is an integer, determine if G has a dominating set D of k vertices (i.e., a set D of k vertices in G such that every vertex in G is either in D or is adjacent to a vertex in D).

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 $\pi(G)$. Inductively, a vertex v is in layer-i, $i > 1$, if v is on the boundary of the unbounded region of the embedding induced from $\pi(G)$ by deleting all layer-j vertices for all $1 \leqslant j \leqslant i-1$. The embedding $\pi(G)$ is *q-outerplanar* if it has at most q layers.

Theorem 3.5[41]. *Suppose that a graph G has a q-outerplanar embedding $\pi(G)$. Then a minimum dominating set D of G can be constructed in time $O(c^q|G|)$, where c is a constant.*

Theorem 3.5 is obtained by applying the technique of graph separators[41]: 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 (hence, the boundary vertices of S make a separator that separates the rest of the vertices in S from the graph G). A *trivial slice* is simply an edge in G. Two slices S_1 and S_2 can be "merged" into a larger slice if the right boundary of S_1 is identical to the left boundary of S_2. Baker[41] presented a linear time algorithm to show how a q-outerplanar embedding $\pi(G)$ is decomposed into slices and how the slices, starting from trivial slices, are recursively merged to reconstruct the original graph G.

Note that the minimum dominating set D uniquely determines a "configuration" C_S for the boundary vertices in a slice S, consisting of a status for each boundary vertex v (represented by a color): (1) *black*: v is in D; (2) *gray*: v is not in D and is adjacent to a vertex in $S \cap D$; and (3) *white*: v is not in D and is adjacent to a vertex in $D - S$. Since D is a minimum dominating set and the boundary vertices of S separate S from the rest of the

graph G, $D \cap S$ must be a minimum set in S such that coloring all vertices in $D \cap S$ "black" induces the configuration C_S (we will simply say "the set $D \cap S$ achieves the configuration C_S"). In fact, it is easy to see that any minimum set D_S in S that achieves the configuration C_S can replace $D \cap S$, and $D' = D - S + D_S$ also makes a minimum dominating set for the graph G. Therefore, what is important is not the set $D \cap S$ itself but the configuration C_S of the boundary vertices of the slice S. Hence, we can represent the dominating set D restricted to the slice S, i.e., the set $D \cap S$, by an "augmented configuration", which is the configuration C_S plus the size $b(C_S)$ of a minimum set in S achieving C_S.

Based on this observation, a minimum dominating set D for the graph G can be constructed using the following dynamic programming process. For each slice S, we keep the set A_S of all augmented configurations for S. Since the slice S has at most $2q$ boundary vertices and each vertex can be colored by one of the three colors, the set A_S contains at most 3^{2q} augmented configurations. For a trivial slice S, the value $b(C_S)$ can be computed directly for each configuration C_S. To compute the augmented configurations for a larger slice S, suppose that the slice S is obtained from two smaller slices S_1 and S_2 by merging the right boundary of S_1 and the left boundary of S_2, and that the augmented configurations for S_1 and S_2 have been constructed. Then we can construct all augmented configurations for the slice S by enumerating all consistent pairs of augmented configurations of S_1 and S_2. Note that there are at most $3^{2q} \cdot 3^{2q} = 81^q$ pairs of augmented configurations for S_1 and S_2. Therefore, the set of augmented configurations for the slice S can be constructed in time $O(81^q)$. Using this dynamic programming process plus Baker's slice decomposition[41], from the q-outerplanar embedding $\pi(G)$ of the graph G, we can construct a minimum dominating set for the graph G in time $O(81^q|G|)$.

Now we consider the problem for general planar graphs. Let (G, k) be an instance for the PLANAR DOMINATING SET problem. By applying the kernelization algorithm[33], we can assume without loss of generality that the number n of vertices in G is bounded by $335k$. Let $\pi(G)$ be a planar embedding of G. Partition the layers of $\pi(G)$ into "chunks", each consisting of $\sqrt{k}+1$ consecutive layers in $\pi(G)$. By taking three consecutive layers in each trunk, we get a set T_0 of vertices in G. Note that there are $(\sqrt{k}+1)/3$ such vertex sets, which are all disjoint. Therefore, we can assume that the one T_0 we pick satisfies $|T_0| \leqslant 335k/((\sqrt{k}+1)/3) = O(\sqrt{k})$. Similar to the case for slices, a minimum dominating set D for the graph G determines a unique color assignment to the vertices in T_0. Note that this color assignment must be "feasible" and not assign the color *white* to any vertex in the middle layer in any three consecutive layers in T_0. Therefore, the dominating set D can be constructed by enumerating all feasible color assignments to the vertices in T_0. For each feasible color assignment, since the statuses of all vertices in the middle layer of any three consecutive layers in T_0 are completely decided (i.e., for each v of such vertices, we not only know whether v is in D, but also know, in case v is not in D, which vertex in D is adjacent to v), we can remove these vertices from the consideration. Note that these removed vertices make a separator of the graph G that separates G into components, each having a $\sqrt{k}$-outerplanar embedding (the vertices in the first and the last layers of each component have pre-assigned colors). Now using Theorem 3.5, we can construct a smallest dominating set for each component G' that achieves the colors in the first and the last layers of G', in time $O(81^{\sqrt{k}}|G'|)$. Combining these smallest dominating sets for the components and the *black* vertices in T_0 gives a smallest dominating set for the graph G that is consistent with the color assignment in T_0. By trying all feasible color assignments to the vertices in T_0 (there are at most $3^{|T_0|} = 2^{O(\sqrt{k})}$ such feasible color assignments), we will get a minimum dominating set for the graph G. In summary, the above algorithm runs in time $2^{O(\sqrt{k})}|G|$ and solves the PLANAR DOMINATING SET problem.

The same techniques can be applied to solve many other parameterized problems on planar graphs, such as PLANAR VERTEX COVER and PLANAR INDEPENDENT SET. Note that the running time of the algorithm is actually "subexponential" in terms of the parameter k. More general, the technique can be used to develop fixed-parameter algorithms for the WEIGHTED SATISFIABILITY problem on planar circuits of constant depths, which implies parameterized algorithms of similar running time for most parameterized problems on planar graphs[43].

The above discussion can be rephrased based on the concept of the TREE DECOMPOSITIONS of graphs of small treewidth[44]. As pointed out by Eppstein[45], Baker's slice decomposition on a q-outerplanar embedding of a graph G is actually a construction of a tree decomposition of treewidth $O(q)$ for the graph. Moreover, the dynamic programming process based on the slice decomposition also resembles the standard dynamic programming technique for solving problems on graphs of small treewidth[46]. In particular, the standard dynamic programming technique for graphs of small treewidth, plus the vertex coloring method described above, gives an $O(3^h|G|)$ time algorithm for the DOMINATING SET problem on graphs with tree decompositions of treewidth bounded by h[47]. Finally, based on the layered structures of a planar embedding of a planar graph, it can be shown that a planar graph that has a dominating set of k vertices has a tree decomposition of treewidth bounded by $6\sqrt{34}\sqrt{k}+8$, and the tree decomposition can be constructed in polynomial time[47]. Combining these two results gives an algorithm of running time $O(3^{6\sqrt{34}\sqrt{k}}|G|)$ for the PLANAR DOMINATING SET problem[47]. Continuous improvements over this algorithm have appeared[48,49]. The fastest known algo-

rithm for this problem has its running time bounded by $O(2^{15.13\sqrt{k}}k+n^3)$ and was based on the concept of graph branch decompositions[48].

3.4 Graph Minor Theory and Others

There are other effective techniques developed in the study of parameterized algorithms for fixed-parameter tractable problems, such as the techniques of integer linear programming and color-coding and hashing. Using the technique of color-coding and hashing, Alon, Yuster, and Zwick[50] were able to show that the LONGEST PATH problem (determining if a given graph G has a simple path of length k) is fixed-parameter tractable. Based on a technique called *greedy localization*, efficient algorithms for the problems 3D-MATCHING, 3-SET PACKING, and SET SPLITTING have been developed[51,52,53]. We recommend Niedermeier's recent thesis[54] for a more comprehensive and detailed description.

Before we close this section, we briefly describe a research area in graph theory, the *graph minor theory*, which seems to be of great theoretical importance for the study of fixed-parameter tractability of parameterized graph problems.

Let $e=[u,v]$ be an edge in a graph G. By *contracting the edge* e, we mean that we remove the edge e then identify the two ends u and v. We say that a graph H is a *minor* of another graph G if H can be obtained from a subgraph of G by a series of edge contraction operations. We say that a class $\mathcal{F}$ of graphs is *minor closed* if for each graph G in $\mathcal{F}$, all minors of G are also in $\mathcal{F}$. Let $\mathcal{F}$ be a minor closed graph class. A graph G is a *minimal forbidden minor* for $\mathcal{F}$ if G is not in $\mathcal{F}$ but every proper minor of G is in $\mathcal{F}$. The set of all minimal forbidden minors for the graph class $\mathcal{F}$ is called the *set of minimal forbidden minors* for $\mathcal{F}$.

The study of graph minors, initialized by Robertson and Seymour[55−57], has induced significant progress in the research in graph theory. For example, the concept of graph tree decompositions we described in Subsection 3.3 was a product of this research. In particular, we have the following result (formerly known as "Wagner's Conjecture").

Theorem 3.6[55]**.** *Any minor closed graph class has a finite set of minimal forbidden minors.*

Theorem 3.6 is a significant generalization of the famous Kuratowski-Theorem for planar graphs. Indeed, the class of planar graphs is minor closed, and according to Kuratowski-Theorem, the set of minimal forbidden minors for the class of planar graphs consists of two graphs: the complete graph K_5 and the complete bipartite graph $K_{3,3}$.

As indicated by Fellows and Langston[58], Theorem 3.6 has a significant impact in deriving polynomial time computability for many graph problems. Similarly, from the theoretical point of view, Theorem 3.6 is very significant in deriving the fixed-parameter tractability for many parameterized graph problems. We consider as an example the GRAPH GENUS problem (for a given pair (G,k), determine if the minimum genus of the graph G is bounded by k).

For a fixed integer k, the class of graphs whose minimum genus is bounded by k is clearly minor closed: indeed, if H is a minor of a graph G of minimum genus bounded by k, then an embedding of genus bounded by k for H can be obtained from an embedding of genus bounded by k for the graph G by a series of edge deletions and edge contractions. According to Theorem 3.6, we have:

Theorem 3.7[56]**.** *For a fixed integer $k \geqslant 0$, the set $\mathcal{E}_k$ of minimal forbidden minors for the class of graphs of minimum genus bounded by k is finite.*

In order to describe the parameterized algorithm for GRAPH GENUS, we need another result by Robertson and Seymour.

Theorem 3.8.[57]**.** *Let H be a fixed graph. Then there is an algorithm that for a given graph G decides in time polynomial in $|G|$ whether H is a minor of G.*

Now for an instance (G,k) of the GRAPH GENUS problem, we first construct the set $\mathcal{E}_k$ of minimal forbidden minors for the graphs whose minimum genus is bounded by k. The set $\mathcal{E}_k$ is finite and depends only on k. Thus, the set $\mathcal{E}_k$ can be constructed in time $f_1(k)$ for a recursive function f_1 by checking all the graphs of size bounded by a function of k. The number of graphs in $\mathcal{E}_k$ is also bounded by a function $f_2(k)$ of k. Now for each graph H in $\mathcal{E}_k$, we use Theorem 3.8 to test if H is a minor of G, which takes time polynomial in $|G|$. If any graph in $\mathcal{E}_k$ is a minor of G, then clearly (G,k) is a no-instance of GRAPH GENUS. Otherwise, (G,k) is a yes-instance of GRAPH GENUS. The running time of this algorithm is bounded by $O(f_3(k)|G|^c)$, where f_3 is a recursive function and c is a fixed constant. This shows that the GRAPH GENUS problem is fixed-parameter tractable.

The algorithm given above is only of theoretical interest. In fact, the number of graphs in the set $\mathcal{E}_k$ is an impractically large function of k.

4 How Hard Can $W[1]$ Be: Lower Bound Techniques

The establishment of NP-completeness theory enables us to identify intractable problems, i.e., NP-hard problems, whose intractability is not derived by formal mathematical proofs, but is from the extensive computational experience and practice. Now it has become widely accepted that NP-hard problems cannot be solved by any computation models practically. Therefore, the NP-hardness of a problem provides an effective computational lower bound for the problem.

The theory of fixed-parameter intractability was developed based on the similar philosophy. Our lower bound techniques are based on the following working hypothesis:

Working Hypothesis I. *FPT* $\neq W[1]$, *i.e., no* $W[1]$*-hard problem is fixed-parameter tractable.*

In this section, we first discuss how solid our working hypothesis is. We then describe techniques that enable us to derive computational lower bounds for many parameterized problems, based on our working hypothesis. In particular, we will see that for certain parameterized problems, we can derive much stronger computational lower bounds based on our working hypothesis than those one may derive based on NP-completeness theory.

4.1 How Solid Our Working Hypothesis Is

A number of $W[1]$- or $W[2]$-complete problems, such as the well-known INDEPENDENT SET, DOMINATING SET, and WEIGHTED CNF-SAT problems, have been the main targets for the research in exact algorithms for the last three decades (see, for example, [59]). The fact that no one has been able to develop a fixed-parameter tractable algorithm for any of these problems provides the first evidence supporting our working hypothesis.

We say that a problem is *subexponential time solvable* if it can be solved in time $2^{o(n)}$, where n is the length of the input. Our working hypothesis has a close connection to the recent study of subexponential time computability of NP-hard problems. In the study of approximability of NP-hard problems, Papadimitriou and Yannakakis[60] introduced the class SNP, which contains all search problems expressible by second-order existential formulas whose first-order part is universal. The class SNP contains many well-known NP-hard optimization problems, and forms the core for the approximation class APX, which contains all NP optimization problems with constant-ratio polynomial-time approximation algorithms[2]. Impagliazzo and Paturi[61] introduced a slight generalization of the class SNP, the *size-constrained SNP*, and showed that the following problems are also in the size-constrained SNP: CNF q-SAT for $q \geqslant 3$, q-COLORABILITY for $q \geqslant 3$, q-SET COVER for $q \geqslant 2$, INDEPENDENT SET, CLIQUE, and VERTEX COVER.

Theorem 4.1. *Unless all size-constrained SNP problems are solvable in subexponential time,* $FPT \neq W[1]$.

Proof. It is known that $FPT = W[1]$ would imply that the problem CNF 3-SAT can be solved in subexponential time[62]. According to Impagliazzo and Paturi[61], CNF 3-SAT is SERF-complete for the class size-constrained SNP, in the sense that if CNF 3-SAT is subexponential time solvable then all size-constrained SNP problems are also subexponential time solvable. The theorem follows directly from these two facts. □

The development of improved algorithms for certain size-constrained SNP problems, such as CNF 3-SAT, 3-COLORABILITY, INDEPENDENT SET, CLIQUE, and VERTEX COVER, has been an active research subarea in computer science for many years[36]. Each of these problems has passed through many rounds of improvement, and has their best algorithm of running time of the form $O(c^n)$, for a constant $c > 1$. Further improvement on the algorithm to even slightly decrease the constant c seems to be very difficult and involved, and requires smart new ideas. Therefore, it would be a quite surprising breakthrough that all size-constrained SNP problems, including all the above problems, are solvable in subexponential time. This fact has become increasingly important recently in the study of parameterized complexity, which has allowed us to make our second working hypothesis as follows.

Working Hypothesis II. *Not all size-constrained SNP problems are subexponential time solvable.*

Note that Working Hypothesis II implies Working Hypothesis I. A subclass of the class $W[1]$, MINI[1], has been introduced recently[63] that is closely related to Working Hypothesis II. It can be shown that MINI[1] $= FPT$ if and only if all size-constrained SNP problems are subexponential time solvable[63]. Therefore, Working Hypothesis II can be rephrased as MINI[1] $\neq FPT$.

In applications, it has been a commonly accepted practice for researchers in various areas to derive computational lower bounds for their own problems based on the working hypotheses. For example, Papadimitriou and Yannakakis[14] have used Working Hypothesis I to show that the DATABASE QUERY EVALUATION problem is $W[1]$-hard, and hence the problem is intractable even for small query size. Bodlaender *et al.*[64] identified a number of problems in computational biology and showed the hardness of these problems in various levels in the W-hierarchy, thus establishing the intractability of these problems even for small parameter values.

4.2 Intractability of NP-Hard Problems with Small Parameters

According to Working Hypotheses I, no $W[1]$-hard problem is fixed-parameter tractable. Therefore, any parameterized algorithm solving a $W[1]$-hard problem must have its running time taking the form $O(n^{h(k)})$, where $h(k)$ is an unbounded function. Since algorithms of running time larger than $O(n^{10})$ would be considered impractical, the $W[1]$-hardness of a parameterized problem provides strong evidence for the intractability of the problem even for small parameter values.

Many parameterized problems are identified that are hard or complete for various levels of the W-hierarchy. We refer the readers to [65] and [18] for a comprehensive summary. In particular, many well-known NP-hard problems are included in this list: INDEPENDENT SET, CLIQUE, WEIGHTED CNF q-SAT for any fixed constant $q \geqslant 2$, and SET PACKING are $W[1]$-complete, DOMINATING SET, SET COVER, FEATURE SET, and WEIGHTED CNF-SAT are $W[2]$-complete.

We add a few more problems to this list, which were discovered more recently.

Computational biology has become quite popular in recent years. A frequently recurring problem in biology

applications is to find one substring of length n that appears (with a few substitutions) at least once in each of a set of bad strings (such as bacterial sequences) and is not "close" to any substring of length n in each of another set of good strings (such as human and livestock sequences). The problem has various applications in molecular biology such as the design of universal PCR primers, identification of genetic drug targets, and design of genetic probes. In particular, the genetic drug target identification problem searches for a sequence of genes that is close to bad genes (the target) but far from all good genes (to avoid side-effects). This problem can be formulated as follows. Consider all strings in 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, $|s_1| \leqslant |s_2|$, 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 $|s_1| < |s_2|$, 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): for a given pair (x, k), where x is a tuple (n, S_b, S_g, d_b, d_g), with integers n, d_b, and d_g, $d_b \leqslant d_g$, $S_b = \{b_1, \ldots, b_{n_b}\}$ is the set of bad strings, $|b_i| \geqslant n$, and $S_g = \{g_1, \ldots, g_{n_g}\}$ is the set of good strings, $|g_j| = n$, the parameter is given as $k = d_b + d_g$, determine if there is a string s of length n such that $D(s, b_i) \leqslant d_b$ for all $1 \leqslant i \leqslant n_b$, and $D(s, g_j) \geqslant d_g$ for all $1 \leqslant j \leqslant n_g$.

The DSSP problem is NP-hard[66]. Recently, it has been shown[67] that DSSP is $W[1]$-hard. By a different fpt-reduction from DOMINATING SET, we can improve this result and prove:

Theorem 4.2[68]. *The* DSSP *problem is* $W[2]$*-hard.*

Therefore, even for small string matching errors d_b and d_g, the DSSP problem remains intractable.

Our second problem is motivated by applications in VLSI design, communication networks, and data mining[63], which for a given weighted graph, tries to remove an edge subset of small weight so that the graph is split into many pieces. The problem is formally stated as follows.

GRAPH CUT: for a given pair (x, k), where k is the parameter, $x = (G, w)$, G is a weighted graph and w is an integer, determine if there is an edge subset C of total weight bounded by w such that removing the edges in C splits the graph into at least k nonempty connected components.

The GRAPH CUT problem is NP-hard[69]. A classic result for the problem is an algorithm of running time $O(n^{k^2})$ by Goldschmidt and Hochbaum[69]. The algorithm is obviously not practical even for $k = 3$. It therefore is natural to ask if one can hope a significant improvement on the algorithm so that it will become practical at least for small parameter values k.

Theorem 4.3[63]. *The* GRAPH CUT *problem is* $W[1]$*-hard.*

Therefore, again, GRAPH CUT will remain intractable even for small parameter values k.

4.3 Intractability of Problems That Are Not NP-Hard

A nice contribution of the theory of parameterized complexity is that it allows us to derive the intractability for certain problems that may not be derivable based on the NP-completeness theory. Indeed, $W[1]$-hardness does not necessarily imply NP-hardness. Therefore, a problem that may not be NP-hard can still be $W[1]$-hard, thus establishing the intractability for the problem.

We start by considering the V-C DIMENSION problem, which is defined as follows: given a pair $(\mathcal{C}, k)$, where $\mathcal{C}$ is a collection of subsets in a universal set U, decide whether there is a subset S of k elements in U such that for every subset T of S, there is a subset $C_T \in \mathcal{C}$ satisfying $S \cap C_T = T$. The cardinality of the largest such subset S is called the *V-C dimension* of the collection $\mathcal{C}$, which is a measure of the "variability" of $\mathcal{C}$ and is a reasonably precise estimate of the complexity of learning $\mathcal{C}$ when $\mathcal{C}$ is thought of as a class of concepts to be learned[16].

Close inspection reveals that the V-C dimension of a collection $\mathcal{C}$ is always bounded by $\log |\mathcal{C}|$ (see [17]). Therefore, an instance $(\mathcal{C}, k)$ of the V-C DIMENSION problem will become nontrivial only when $k \leqslant \log |\mathcal{C}|$. As a consequence, the problem can be solved in time $O(n^{O(\log |\mathcal{C}|)})$ by enumerating all subsets of k elements in U. This shows that the problem is unlikely to be NP-hard.

Theorem 4.4[18]. *The* V-C DIMENSION *problem is* $W[1]$*-complete.*

Therefore, even the V-C DIMENSION problem seems easier than NP-complete problems, according to our working hypothesis, the problem is still intractable even for small parameter values.

We give another example in computational biology. A typical instance of the MOTIF FINDING problem is stated as follows: given 20 sample DNA sequences, find an (l, d)-motif (i.e., a pattern of length l that matches a subsequence in each given sequence with at most d mismatches allowed). Pevzner and Sze[19] proposed a graph theoretical approach to the MOTIF FINDING problem, and considered the following graph problem:

CLIQUE IN k-GRAPHS: given a pair (G, k), where G is a k-partite graph (i.e., the vertices of G are partitioned into k groups and no two vertices in the same group are adjacent), decide if G has a clique of size k.

In particular, the above MOTIF FINDING problem can be approached by solving the CLIQUE IN 20-GRAPHS problem. Clearly, the CLIQUE IN 20-GRAPHS problem can be solved in time $O(n^{20})$ by examining each subset of 20 vertices in the given graph, hence it is polynomial time solvable. On the other hand, all proposed algorithms and implementations for the problem so far are extremely time consuming, taking a few days or even a few weeks[20].

Theorem 4.5[20]. *The* CLIQUE IN k-GRAPHS *problem is* $W[1]$*-complete.*

Theorem 4.5 explains why the approach in [19] cannot be very efficient: the CLIQUE IN k-GRAPHS problem is intractable even for small parameter values k. In particular, for the parameter value $k = 20$, which corresponds to solving the original MOTIF FINDING problem, one may not expect a very efficient algorithm for the problem.

We close this subsection with another example. Consider the following problem:

PRECEDENCE CONSTRAINED PROCESSOR SCHEDULING (PCPS): given a pair (T, k), where T is a set of jobs of unit length with a partial order defined on the jobs, and a deadline D, decide if there is a scheduling of the jobs on k identical processors that is consistent with the job partial order and finishes all jobs within D time units.

When the number k of processors is fixed to 2, the PCPS problem is solvable in polynomial time. However, it has been an open problem for nearly 30 years whether for any fixed number $k \geqslant 3$ of processors, the problem is NP-hard[1].

Theorem 4.6[70]. *The problem* PCPS *is* $W[2]$*-hard.*

By Working Hypothesis I, therefore, we can obviate the tricky question whether PCPS is NP-hard, but are still able to conclude the intractability for the problem: no matter whether the problem is NP-hard for a fixed number k of processors, when k is not a very small constant, the problem becomes intractable.

4.4 Strong Computational Lower Bounds

On the surface, the $W[1]$-hardness of a parameterized problem implies that any algorithm solving the problem must have its running time taking the form of $O(n^{h(k)})$, where $h(k)$ is an unbounded function. However, this does not entirely exclude the possibility that a $W[1]$-hard problem becomes tractable for small parameter values. For instance, if the problem is solvable in time $O(n^{\log\log k})$, then the problem is feasible even for parameter values up to $k = 1,000$.

Very recent research has shown that based on the working hypotheses, for many known $W[1]$-hard parameterized problems, we can actually derive much stronger computational lower bounds.

Let us consider the $W[2]$-complete problem WEIGHTED CNF-SAT: "given a CNF formula F and an integer k, decide if the formula F has a satisfying assignment of weight k." The problem can be simply solved by enumerating all weight-k assignments to the n input variables in the formula F. This process takes time $O(n^k m^2)$, where $m = |F|$ is the instance size. Thus, besides the parameter k and the instance size m, the number n of input variables in F has played an important role in determining the computational complexity of the problem. The above simple algorithm runs in exponential time, but the exponential factor is in terms of the number n of input variables in F, instead of the instance size m. Note that the instance size m can be of the order 2^n.

Each instance (F, k) of WEIGHTED CNF-SAT can be regarded as a search problem, in which we need to select k elements from a searching space consisting of a set of n input variables, and assign them value 1 so that the formula F is satisfied. Many well-known computational problems have similar formulations. For example, for the SET COVER problem (given a collection $\mathcal{C}$ of subsets in a universal set U, and an integer k, decide whether there is a subcollection of k subsets in $\mathcal{C}$ whose union is equal to U), the searching space is $\mathcal{C}$; for the HITTING SET problem (given a collection $\mathcal{C}$ 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 $\mathcal{C}$), the searching space is U.

For many graph problems, the searching space seems naturally the entire set of the vertices in the graph. In this case, the size n of the searching space is polynomially related to the instance size m. For certain graph problems, however, a polynomial time preprocessing on the input instances may significantly reduce the size of the searching space. For example, for finding a vertex cover of k vertices in a graph of n vertices, where n can be much larger than k, the kernelization algorithm described in Subsection 3.1 can reduce the searching space size to $2k$. For finding a dominating set of k vertices in a graph, a polynomial time algorithm has been proposed in [68] that identifies a subset of vertices from which the k vertices in the dominating set can be selected.

We will concentrate in this subsection on parameterized problems that seek a subset in a searching space satisfying certain properties. For most problems in our consideration, the searching space can be easily identified. For some other problems, such as DOMINATING SET, the searching space can be identified by a polynomial time preprocessing. If none of these is the case, we then simply pick the entire input instance as the searching space. In the remaining discussion in this subsection, for each problem instance, we will denote by n the size of the searching space and m the size of the instance.

Theorem 4.7[15]. *If any of the following problems can be solved in time* $O(n^{o(k)}p(m))$ *for a polynomial* p*, then* $W[1] = FPT$: WEIGHTED CNF-SAT, DOMINATING SET, HITTING SET, *and* SET COVER.

Theorem 4.7 shows that a parameterized algorithm solving any of the problems in the theorem must have its running time of the order $n^{\Omega(k)}p(m)$ for any polynomial p. This computational lower bound is asymptotically tight, in the sense that each of these problems can be solved by a trivial algorithm of running time $O(n^k m^2)$. For example, for the WEIGHTED CNF-SAT problem, on an input (F, k), we can simply pick every subset of k input variables in the formula F, assign these k variables value 1 and the rest $n - k$ variables value 0, and check if this assignment satisfies the formula F.

Theorem 4.7 can be further strengthened.

Theorem 4.8[68]. *If any of the parameterized problems in Theorem* 4.7 *can be solved in time* $f(k)n^{o(k)}p(m)$

for any function f and any polynomial p, then $W[1] = FPT$.

Therefore, for any function $f(k)$ and any polynomial $p(m)$, a parameterized algorithm solving any of the problems in Theorem 4.7 still needs to take time of order $f(k)n^{\Omega(k)}p(m)$. This result has the following interesting interpretation. All problems in Theorem 4.7 have trivial algorithms of running time $O(n^k m^2)$ by the straightforward method of exhaustive enumerations. 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. 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[40,71] in order to further improve the computational time needed to solve the problems. Theorem 4.8, however, provides strong evidence that the power of this approach is quite limited in principle. The lower bound $f(k)n^{\Omega(k)}p(m)$ for any function f and for any polynomial p in the theorem indicates that, in principle, *no* local structure manipulation running in polynomial time or in time depending only on the parameter value k will obviate the need for the exhaustive enumerations.

Theorem 4.9[68]. *Let $\mu(n) = O(n^\epsilon)$ be any nondecreasing and unbounded function, where $0 < \epsilon < 1$ is a constant. Then for each of the problems in Theorem 4.7, there are two constants c_1 and c_2, $c_1 \leqslant c_2$, such that the problem cannot be solved in time $O(n^{o(k)}p(m))$ for any polynomial p, even if we restrict the parameter values k to be $c_1\mu(n) \leqslant k \leqslant c_2\mu(n)$, unless $W[1] = FPT$.*

Theorem 4.9 shows that the problems in Theorem 4.7 are not only intractable in general, they are actually intractable for *every* parameter value k. In particular, any special range of the parameter values would not decrease the complexity of the problems. Thus, for any polynomial p, finding a satisfying assignment of weight $\log n$ for a formula will take time of order at least $n^{\Omega(\log n)}p(m)$, and finding a satisfying assignment of weight $\sqrt{n}$ for a formula will take time of order at least $n^{\Omega(\sqrt{n})}p(m)$.

All problems in Theorem 4.7 are $W[2]$-complete. For certain $W[1]$-complete problems, we have similar results. Actually, the results are even stronger: the lower bounds are now in terms of the instance size m. On the other hand, instead of using Working Hypothesis I, we need to use the stronger Working Hypothesis II.

Theorem 4.10[15]. *If any of the following problems can be solved in time $O(m^{o(k)})$, then all size-constrained SNP problems are solvable in subexponential time:* WEIGHTED CNF q-SAT *for any constant $q \geqslant 2$,* CLIQUE, *and* INDEPENDENT SET.

Theorem 4.11[68]. *If any of the parameterized problems in Theorem 4.10 can be solved in time $f(k)m^{o(k)}$ for any recursive function f, then all size-constrained SNP problems are solvable in subexponential time.*

Theorem 4.12[68]. *Let $\mu(m) = O(m^\epsilon)$ be any nondecreasing and unbounded function, where $0 < \epsilon < 1$ is a constant. Then for each of the problems in Theorem 4.10, there are two constants c_1 and c_2, $c_1 \leqslant c_2$, such that the problem cannot be solved in time $m^{o(k)}$, even if we restrict the parameter values k to be $c_1\mu(m) \leqslant k \leqslant c_2\mu(m)$, unless all size-constrained SNP problems are solvable in subexponential time.*

We point out that the list of parameterized problems studied in this subsection gives sample problems with strong computational lower bounds. This list is far from being exhaustive. The computational lower bounds were first derived for the problems WCS$^*[t]$ and WCNF-2SAT$^-$[68]. The lower bounds for the problems in the list were obtained by a refined fpt-reduction, the *linear fpt-reduction*, from WCS$^*[t]$ and WCNF-2SAT$^-$[68]. Therefore, in general, the computational lower bounds will hold for any parameterized problem Q if either WCS$^*[t]$ or WCNF-2SAT$^-$ (or any of the problems listed in this subsection) can be linearly fpt-reducible to Q.

4.5 Computational Lower Bounds for Certain FPT Problems

Our computational lower bounds so far were derived from the $W[1]$-hardness of parameterized problems, using our working hypotheses. In this subsection, we show that computational lower bounds for certain fixed-parameter tractable problems can also be derived using our working hypotheses.

We start with the parameterized problem VERTEX COVER. Note that all existing parameterized algorithms for VERTEX COVER have their running time of the form $O(2^{ck}p(n))$ for a constant $c > 0$, where n is the number of vertices in the input graph. Improved algorithms for the problem have been focused on the improvement on the constant c. Theoretically, we are interested in knowing how far we can go with the constant c. Can the constant c be arbitrarily close to 0?

Theorem 4.13[72]. *The* VERTEX COVER *problem is solvable in time $O(2^{o(k)}p(n))$ for a polynomial p if and only if all size-constrained SNP problems are solvable in subexponential time.*

Cai and Juedes[72] have shown that there are a number of fixed-parameter tractable problems for which the lower bound in Theorem 4.13 also holds. These problems include MAX-SAT, BOUNDED-DEGREE VERTEX COVER, BOUNDED-DEGREE INDEPENDENT SET, and BOUNDED DEGREE DOMINATING SET (i.e., the corresponding problems on graphs whose vertex degree is bounded by a constant). These results show that for each of these problems, there is a constant $c_0 > 0$ such that the problem cannot be solved in time $O(2^{c_0 k}p(n))$ for any polynomial p.

This study has also led to computational lower bounds for problems on planar graphs.

Theorem 4.14[72]. *If the* PLANAR VERTEX COVER

problem can be solved in time $O(2^{o(\sqrt{k})}p(n))$ for a polynomial p, then all size-constrained SNP problems are solvable in subexponential time.

The lower bound in Theorem 4.14 can be extended to other problems on planar graphs, such as PLANAR INDEPENDENT SET and PLANAR DOMINATING SET[72]. Note that these computational lower bounds are tight since upper bounds of the same order for the problems have also been developed using the technique of graph separators (see Subsection 3.3).

5 Fixed-Parameter Tractability and Approximability

One of the most active research areas currently in theoretical computer science is the study of approximation algorithms for NP-hard optimization problems. Research in parameterized complexity theory has shown that fixed-parameter tractability and approximability of optimization problems are closely related. In particular, the study of fixed-parameter intractability has provided new techniques and tools for the study of inapproximability of optimization problems. We first give a brief review on the terminologies in approximation algorithms. The readers are referred to [2] for more detailed definitions and more comprehensive discussions.

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

(3) $f_Q(x, y)$ is the objective function mapping a pair $x \in I_Q$ and $y \in S_Q(x)$ to a non-negative integer. The function f_Q is computable in polynomial time;

(4) $opt_Q \in \{\max, \min\}$. Q is called a *maximization problem* if $opt_Q = \max$, and a *minimization problem* if $opt_Q = \min$.

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

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

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

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

An NP optimization problem Q has a *polynomial time approximation scheme* (PTAS) if there is an algorithm A_Q that takes a pair $\langle x, \epsilon\rangle$ 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 $|x|$. Finally, an NP optimization problem Q has a *fully polynomial time approximation scheme* (FPTAS) if it has a PTAS algorithm A_Q such that the running time of A_Q is bounded by a polynomial of $|x|$ and $1/\epsilon$.

Definition 7. *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*

$$Q_{\geqslant} = \{(x, k) \mid x \in I_Q \text{ and } opt_Q(x) \geqslant k\};$$

(2) *if Q is a minimization problem, then the parameterized version of Q is defined as*

$$Q_{\leqslant} = \{(x, k) \mid x \in I_Q \text{ and } opt_Q(x) \leqslant k\}.$$

The above definition offers the possibility for us to study the parameterized complexity of NP optimization problems in terms of their approximability. We start with a simple but useful theorem.

Theorem 5.1. *Suppose that an NP optimization problem $Q = (I_Q, S_Q, f_Q, opt_Q)$ has a PTAS A of running time bounded by $\beta(|x|, 1/\epsilon)$ on input $\langle x, \epsilon\rangle$, where β is a function that is a polynomial of $|x|$ for each fixed $\epsilon > 0$. Then the parameterized version of Q can be solved in time $\beta(|x|, 2k)$.*

Proof. We prove the theorem for the case when Q is a maximization problem. Consider the following parameterized algorithm $A_{\geqslant}$ for the parameterized version $Q_{\geqslant}$ of Q: on an instance (x, k) of $Q_{\geqslant}$, the algorithm $A_{\geqslant}$ first calls the PTAS algorithm A on input $\langle x, 1/(2k)\rangle$ to construct a solution y_0 for x, then reports "yes" if and only if $f_Q(x, y_0) \geqslant k$. Note that if $f_Q(x, y_0) \geqslant k$ then obviously (x, k) is a yes-instance of $Q_{\geqslant}$ since Q is a maximization problem. In case $f_Q(x, y_0) < k$, because the solution y_0 satisfies the relation $opt_Q(x)/f_Q(x, y_0) \leqslant 1 + 1/(2k)$, we have (note that $f_Q(x, y_0) < k$ and $f_Q(x, y_0)$ is an integer)

$$\begin{aligned} opt_Q(x) &\leqslant f_Q(x, y) + f_Q(x, y)/(2k) \\ &\leqslant k - 1 + 1/2 = k - 1/2 < k. \end{aligned}$$

Therefore, the algorithm $A_{\geqslant}$ correctly determines the membership of the parameterized problem $Q_{\geqslant}$. Moreover, the running time of the algorithm $A_{\geqslant}$ is clearly

bounded by $\beta(|x|, 2k)$. □

5.1 FPTAS, SNP and FPT

Recall that an FPTAS algorithm for an NP optimization problem Q is an approximation algorithm A for Q that on input $\langle x, \epsilon\rangle$ produces a solution y to x in time polynomial in $|x|$ and $1/\epsilon$ such that the approximation ratio of the solution y is bounded by $1+\epsilon$. By Theorem 5.1, we get

Theorem 5.2[73]. *An NP optimization problem Q with an FPTAS is fixed-parameter tractable.*

Theorem 5.2 immediately includes a number of kapnsack-type problems and scheduling problems[1] into the class FPT. On the other hand, it also provides a tool for excluding certain NP optimization problems from the class FPTAS: according to Working Hypothesis I, an NP optimization problem whose parameterized version is $W[1]$-hard would have no FPTAS.

More recent research[43] has shown that under a very general constraint, the approximation class FPTAS can be precisely characterized in terms of parameterized complexity. To describe this result, we first notice that 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 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", i.e., a solution to the input instance, to support the conclusion (see [43] for more detailed discussions).

Definition 8. *An NP optimization problem Q is efficiently fixed-parameter tractable (efficient-FPT) if its parameterized version is solvable in 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 9. *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 $d \geqslant 1$, $x_d = g_1(x,d)$ is an instance of Q such that $|x_d| \leqslant q(|x|)$ and $|opt_Q(x_d) - opt_Q(x)/d| \leqslant q(|x|)$; 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 $|f_Q(x_d, y_d) - f_Q(x,y)/d| \leqslant q(|x|)$.

As shown in [43], 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 $g_2(x_d, y_d) = y_d$.

Theorem 5.3[43]. *Let Q be a scalable NP optimization problem. Then Q has an FPTAS if and only if Q is efficient-FPT.*

In comparison to the previous characterizations of FPTAS[74–76], Theorem 5.3 seems to have the advantage of being more general and more constructive (see [43] for detailed discussions).

Finally, we mention that Theorem 5.2 can be extended to larger approximation classes. Recall that the class SNP introduced by Papadimitriou and Yannakakis[60] has been an important approximation class in the study of approximability of NP optimization problems. Another important approximation class MIN $F^+\Pi_1$ has been introduced by Kolaitis and Thakur[77] in their study of NP minimization problems.

Theorem 5.4[73]. *All NP optimization problems in the class SNP or in the class MIN $F^+\Pi_1$ are fixed-parameter tractable.*

Again, Theorem 5.4 includes a large number of NP optimization problems in the class FPT, and provides new tools for excluding certain NP optimization problems from the classes SNP and MIN $F^+\Pi_1$.

5.2 Efficient PTAS and FPT

Since the breakthrough research on PCP theory[78], approximation algorithms, in particular the study of polynomial time approximation schemes for NP optimization problems, have been probably one of the most exciting research areas in theoretical computer science. New computational lower bounds have been established and new PTAS algorithms have been developed for a large number of NP optimization problems[2].

By the definition, a PTAS algorithm for an NP optimization problem Q is an approximation algorithm A for Q such that on an input pair $\langle x, \epsilon\rangle$, where x is an instance of Q and $\epsilon > 0$, the algorithm A produces a solution y to x of approximation ratio bounded by $1+\epsilon$, and that for any fixed $\epsilon > 0$, the running time of the algorithm A is bounded by a polynomial of $|x|$. Therefore, the running time of the algorithm A can be either of the form $O(f(1/\epsilon)n^c)$ where f is a function and c is a constant, or of the form $O(n^{h(1/\epsilon)})$ where h is a function of $1/\epsilon$. For example, Baker's PTAS for PLANAR INDEPENDENT SET[41] has its running time bounded by $O(2^{3/\epsilon}n)$, and the PTAS by Deng *et al.* for the DSSP problem[66] has its running time of the form $O(n^{1/\epsilon^6})$.

Obviously, a PTAS of the latter form is much less efficient than one of the former form. Downey[79] has surveyed a list of PTAS algorithms of the latter form and demonstrated that even for a moderate error bound $\epsilon = 20\%$, these algorithms on their current form are far from being practical.

One might argue that these PTAS algorithms are important theoretically, and that practically, with more efforts and better algorithmic techniques, faster and practical PTAS algorithms will be developed for the problems.

This raises the following important and interesting question for the study of approximation algorithms.

Question: Are there PTAS problems that have no PTAS algorithms whose running time is of the form $O(f(1/\epsilon)n^c)$? If the answer is yes, how do we identify these PTAS problems?

To study this problem more thoroughly, we first introduce a new definition.

Definition 10. *An NP optimization problem Q has an efficient polynomial time approximation scheme (EPTAS) if it admits a polynomial-time approximation scheme whose running time is bounded by $O(f(1/\epsilon)|x|^c)$, where f is a recursive function and c is a constant.*

In particular, an FPTAS problem has EPTAS. By Theorem 5.1, we get immediately

Theorem 5.5[80]**.** *If an NP optimization problem has EPTAS, then it is fixed-parameter tractable.*

Therefore, according to our working hypotheses, any $W[1]$-hard NP optimization problem will have no EPTAS. This, however, does not completely answer the above question since it is not clear at all whether there are NP optimization problems that are $W[1]$-hard and have PTAS.

By carefully examining the formulation proposed by Khanna and Motwani[81] who attempted to characterize the class PTAS by logic problems on planar structures, Cai *et al.*[82] were able to show that a class in Khanna and Motwani's structure is $W[1]$-hard. To explain, we first need a few terminologies. Let $\mathcal{C}$ be a collection of DNF formulas (they may share common input variables). The *bipartite graph $G_{\mathcal{C}}$ associated with $\mathcal{C}$*, with vertex bipartition $V = I \cup F$, is defined as follows. Each input variable in $\mathcal{C}$ corresponds to a vertex in I and each formula in $\mathcal{C}$ corresponds to a vertex in F. There is an edge from a vertex x in I to a vertex f in F if and only if the input variable x appears in the formula f.

Theorem 5.6[81,82]**.** *The following problem has PTAS and is $W[1]$-hard. Therefore, the problem has PTAS but has no EPTAS unless $FPT = W[1]$:*

PLANAR TMIN: *given a collection $\mathcal{C}$ of DNF formulas, with all literals positive and with the associated bipartite graph being planar, construct an assignment of minimum weight that satisfies all formulas in $\mathcal{C}$.*

Therefore, under the working hypotheses in parameterized complexity theory, PLANAR TMIN is such a problem whose $(1+\epsilon)$-approximation is theoretically tractable (i.e., can be constructed in polynomial time) but practically infeasible (i.e., cannot be constructed by practically efficient algorithms). Hence, the study in parameterized complexity theory has provided new techniques to identify those PTAS problems that have no "practically effective" PTAS algorithms.

Finally, we point out that based on the framework of parameterized complexity theory, by enforcing a planar structure on the parameterized classes in the W-hierarchy, it seems possible to structurally characterize most EPTAS problems[43].

5.3 Lower Bounds on the Running Time of PTAS

Let us re-consider the DISTINGUISHING SUBSTRING SELECTION problem (DSSP) (see Subsection 4.2 for the definition).

Theorem 5.7[66]**.** *The DSSP problem has PTAS.*

Strictly speaking, the DSSP problem considered in [66] does not take the standard definition of an NP optimization problem, and the PTAS algorithm given in [66] for DSSP is neither a standard PTAS algorithm. However, by properly re-formulating the problem, we can make the DSSP problem into a standard NP optimization problem and the PTAS algorithm given in [66] will become a standard PTAS algorithm for the standard form (see [83] for details).

Using the techniques of Subsection 4.4, and by a linear fpt-reduction from the DOMINATING SET problem, we can prove:

Theorem 5.8[68]**.** *The DSSP problem is $W[2]$-hard. Moreover, if DSSP can be solved in time $O(f(k)n^{o(k)})$ for any recursive function f, then all size-constrained SNP problems are solvable in subexponential time.*

The following theorem is a corollary of Theorem 5.1.

Theorem 5.9[83]**.** *If an NP optimization problem Q has a PTAS algorithm of running time $O(f(1/\epsilon)n^{o(1/\epsilon)})$ for a recursive function f, then the parameterized version of Q can be solved in time $O(f(2k)n^{o(k)})$.*

Combining Theorems 5.8 and 5.9, we get immediately

Theorem 5.10. *Unless all size-constrained SNP problems are solvable in subexponential time, the DSSP problem has no PTAS of running time $f(1/\epsilon)n^{o(1/\epsilon)}$ for any recursive function f.*

Theorem 5.10 seems the first result in which a specific lower bound is derived on the running time of a PTAS for an NP-hard problem. This result also demonstrates potential applications of parameterized complexity theory in the study of approximation algorithms. In most cases, computational lower bounds and inapproximability of optimization problems are derived based on approximation ratio-preserving reductions[2], by which if a problem Q_1 is reduced to another problem Q_2, then Q_2 is at least as hard as Q_1. In particular, if Q_1 is reduced to Q_2 under an approximation ratio-preserving reduction, then the approximability of Q_2 is at least as difficult as that of Q_1. Therefore, the intractability of an "easier" problem in general cannot be derived using such a reduction from a "harder" problem. On the other hand, our computational lower bound on the DSSP problem was obtained by a linear fpt-reduction from the DOMINATING SET problem. It is well-known that DOMINATING SET has no polynomial time approximation algorithms of

constant ratio[2], while the DSSP problem has PTAS[66]. Thus, from the viewpoint of approximability, DOMINATING SET is much harder than DSSP, and our linear fpt-reduction reduces a harder problem to an easier problem. This hints that our approach for deriving computational lower bounds and the linear fpt-reduction *cannot* be simply replaced by the popular approaches based on approximation ratio-preserving reductions.

6 Conclusions and Further Research

In summary, the theory of parameterized computation and complexity is a recently developed new approach dealing with intractable computational problems. The study has developed new algorithmic techniques for practically solving a large number of computational problems that are intractable in terms of the traditional complexity theory. The study has also offered powerful techniques for deriving strong computational lower bounds for computational problems that may not be derivable based on the traditional NP completeness theory. Before we close this survey, we offer a few directions for further research.

New algorithmic techniques. New algorithmic techniques for the development of parameterized algorithms are highly desired. For some famous fixed-parameter tractable problems, such as VERTEX COVER, we are interested in further improving the existing algorithms. For example, for VERTEX COVER, although it is unlikely to have a parameterized algorithm of time $O(c^k+kn)$ for a constant c arbitrarily close to 1 (see Theorem 4.13), we feel that there should be still some room for further improvement. Note that a slight decrease on the constant c (e.g., from 1.286 to 1.2) will improve the entire running time of the algorithm very significantly[25]. We point out that some powerful techniques developed in the study of general algorithms, such as randomized algorithms and amortized analysis, seem to have great potential for further improvement of parameterized algorithms.

Computational lower bounds. It will be important to identify the $W[1]$-hardness for problems in a variety of applications. Note that such research may have significant impact in the applications, both theoretically and practically (e.g., the DATABASE QUERY EVALUATION problem[14]). Moreover, development of tight computational lower bounds for famous NP-hard parameterized problems (e.g., Subsection 4.4) and proving computational lower bounds on approximation algorithms for NP optimization problems (e.g., Subsection 5.3) are still in their very beginning stages, and should have a great potential for further research.

Study in structural complexity. Many questions have remained open concerning the structures of parameterized computation and complexity. For example, it is still unknown whether the collapse of a particular level in the W-hierarchy (i.e., $W[t] = FPT$ for a particular $t > 0$) would induce the collapse of the entire W-hierarchy. Note that this result holds trivially for most complexity hierarchies proposed and studied in the traditional complexity theory. The recently proposed fixed-parameter intractable class MINI[1][63], which is a subclass of $W[1]$, has turned out to be very important in the study of computational lower bounds for parameterized problems. It is not clear what is the precise relationship between $W[1]$ and MINI[1].

Implementation issues. Parameterized computation and complexity theory has become interesting and important because of its close connection to practical computations in a variety of applications. Besides its theoretical study, we are interested in how practically and how effectively parameterized algorithms work in the real world. Therefore, implementations of the parameterized algorithms in real computer systems to solve computational problems in real world cannot be neglected. Recently, research has started investigating implementations of parameterized algorithms in parallel and distributed systems[8], which has opened new challenging research directions in the study of parameterized computation and complexity theory.

References

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

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

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

[4] Michalewicz Z, Fogel D B. How to Solve It: Modern Heuristics. Springer, Berlin, 2000.

[5] Roth-Korostensky C. Algorithms for Building Multiple Sequence Alignments and Evolutionary Trees [Dissertation]. No. 13550, ETH Zürich, 2000.

[6] Stege U. Resolving Conflicts from Problems in Computational Biology [Dissertation]. No. 13364, ETH Zürich, 2000.

[7] Cai L, Juedes D, Kanj I A. Inapproximability of non NP-hard optimization problems. *Theoretical Computer Science*, 2002, 289: 553–571.

[8] Cheetham J, Dehne F, Rau-Chaplin A, Stege U, Taillon P J. Solving large FPT problems on coarse-granined parallel machines. *J. Computer and System Sciences*, 2003, 67: 691–701.

[9] Lichtenstein O, Pnueli A. Finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symposium on the Principles of Programming Languages*, 1985, pp.97–107.

[10] Henglein F, Mairson H G. The complexity of type inference for higher-order typed lambda calculi. *Journal of Functional Programming*, 1994, 4: 435–477.

[11] Chen J, Kanj I A. Constrained minimum vertex cover in bipartite graphs: Complexity and parameterized algorithms. *Journal of Computer and System Sciences*, 2003, 67: 833–847.

[12] Nesetril J, Poljak S. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 1985, 26: 415–419.

[13] Coppersmith D, Winograd S. Matrix multiplication via arithmetic progression. *J. Symbolic Logic*, 1990, 9: 251–280.

[14] Papadimitriou C H, Yannakakis M. On the complexity of database queries. *Journal of Computer and System Sciences*, 1999, 58: 407–427.

[15] Chen J, Chor B, Fellows M, Huang X, Juedes D, Kanj I, Xia G. Tight lower bounds for certain parameterized NP-hard problems. In *Proc. 19th Annual IEEE Conference on Computational Complexity* (*CCC* 2004), 2004, pp.150–160.
[16] Anthony M, Biggs N. Computational Learning Theory. Cambridge University Press, Cambridge, UK, 1992.
[17] Papadimitriou C H, Yannakakis M. On limited nondeterminism and the complexity of VC dimension. *Journal of Computer and System Sciences*, 1996, 53: 161–170.
[18] Downey R, Fellows M. Parameterized Complexity. Springer-Verlag, 1999.
[19] Pevzner P A, Sze S-H. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proc. 8th International Conf. Intelligent Systems for Molecular Biology* (*ISMB*'00), 2000, pp.269–278.
[20] Sze S H, Lu S, Chen J. Integrating sample-driven and pattern-driven approaches in Motif finding, *Lecture Notes in Computer Science 3240*, 2004, pp.438–449.
[21] Cai L, Chen J, Downey R, Fellows M. On the structure of parameterized problems in NP. *Information and Computation*, 1995, 123: 38–49.
[22] Chen Y, Flum J. Machine characterizations of the classes of the W-hierarchy. *Lecture Notes in Computer Science 2803* (*CSL*'03), 2003, pp.114–127.
[23] Flum J, Grohe M. Describing parameterized complexity classes. *Lecture Notes in Computer Science 2285* (STACS'02), 2002, pp.359–371.
[24] Chen J. Simpler computation and deeper theory: On development of efficient parameterized algorithms. *International Workshop on Parameterized Complexity*, Chennai, India, 2000.
[25] Downey R, Fellows M, Stege U. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics*, Graham R, Kratochvil J, Nesetril J, Roberts F (eds.), *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999, 49: 49–99.
[26] Fellows M. Parameterized complexity: The main ideas and some research frontiers. *Lecture Notes in Computer Science 2223* (ISAAC'01), 2001, pp.291–307.
[27] Cormen T, Leiserson C, Rivest R, Stein C. Introduction to Algorithms. McGraw-Hill, Boston, 2001.
[28] Lovasz L, Plummer M. Matching Theory. North-Holland, Amsterdam, 1986.
[29] Nemhauser G, Trotter L. Vertex packing: Structural properties and algorithms. *Math. Programming*, 1975, 8: 232–248.
[30] Chen J, Kanj I A, Jia W. Vertex cover: Further observations and further improvements. *J. Algorithms*, 2001, 41: 280–301.
[31] Fellows M. Blow-ups, win/win's, and crown rules: Some new directions in FPT. *Lecture Notes in Computer Science* (WG'03), 2003, pp.1–12.
[32] Chor B, Fellows M, Juedes D. An efficient FPT algorithm for saving k colors. *Manuscript*, 2003.
[33] Alber J, Fellows M, Niedermeier R. Polynomial time data reduction for dominating set. *J. ACM*, 2004, 11: 363–384.
[34] Robson J M. Algorithms for maximum independent sets. *J. Algorithms*, 1986, 7: 425–440.
[35] Tarjan R E, Trojanowski A E. Finding a maximum independent set. *SIAM J. Comput.*, 1977, 6: 537–546.
[36] Woeginger G. Exact algorithms for NP-hard problems: A survey. *Lecture Notes in Computer Science 2570*, 2001, pp.185–207.
[37] Niedermeier R, Rossmanith P. A general method to speed up fixed-parameter tractable algorithms. *Inform. Process. Lett.*, 2000, 73: 125–129.
[38] Balasubramanian R, Fellows M R, Raman V. An improved fixed parameter algorithm for vertex cover. *Inform. Process. Lett.*, 1998, 65: 163–168.
[39] Chen J, Liu L, Jia W. Improvement on vertex cover for low-degree graphs. *Networks*, 2000, 35: 253–259.
[40] Niedermeier R, Rossmanith P. Upper bounds for vertex cover further improved. *Lecture Notes in Computer Science 1563* (*STACS*'99), 1999, pp.561–570.
[41] Baker B S. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 1994, 41: 153–180.
[42] Lipton R, Tarjan R. Application of a graph separator theorem. *SIAM Journal on Computing*, 1980, 9: 615–627.
[43] Chen J, Huang X, Kanj I, Xia G. Polynomial time approximation schemes and parameterized complexity. In *Proc. 29th International Symposium on Mathematical Foundations of Computer Science* (*MFCS* 2004), *Lecture Notes in Computer Science 3153*, 2004, pp.500–512.
[44] Bodlaender H L. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 1996, 25: 1305–1317.
[45] Eppstein D. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 2000, 27: 275–291.
[46] Arnborg S. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — A survey. *BIT*, 1985, 25: 2–23.
[47] Alber J, Bodlaender H L, Fernau H, Kloks T, Niedermeier R. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 2002, 33: 461–493.
[48] Fomin F V, Thilikos D M. Dominating sets in planar graphs: Branch-width and exponential speed-up. In *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms* (*SODA*'03), 2003, pp.168–177.
[49] Kanj I, Perkovic L. Improved parameterized algorithms for planar dominating set. *Lecture Notes in Computer Science 2420* (*MFCS*'02), 2002, pp.399–410.
[50] Alon N, Yuster R, Zwick U. Color-coding. *Journal of the ACM*, 1995, 42: 844–856.
[51] Chen J, Friesen D, Kanj I, Jia W. Using nondeterminism to design efficient deterministic algorithms. *Algorithmica*, 2004, 40: 83–97.
[52] Jia W, Zhang C, Chen J. An efficient parameterized algorithm for m-set packing. *Journal of Algorithms*, 2004, 50: 106–117.
[53] Dehne F, Fellows M, Rosamond F. An FPT algorithm for set splitting. *Lecture Notes in Computer Science 2880* (WG'03), 2003, pp.180–191.
[54] Niedermeier R. Invitation to Fixed-Parameter Algorithms [Thesis]. Universitat Tubingen, 2002.
[55] Robertson N, Seymour P D. Graph Minors — A Survey. In Surveys in Combinatorics 1985, Anderson I (ed.), Cambridge Univ. Press, Cambridge, 1985, pp.153–171.
[56] Robertson N, Seymour P D. Graph minors VIII. A Kuratowski theorem for general surfaces. *J. Combin. Theory Ser. B*, 1990, 48: 255–288.
[57] Robertson N, Seymour P D. Graph minors XIII. The disjoint paths problem. *J. Combin. Theory Ser. B*, 1995, 63: 65–110.
[58] Fellows M, Langston M. Nonconstructive tools for proving polynomial-time decidability. *J. ACM*, 1988, 35: 727–739.
[59] *DIMACS Workshop on Faster Exact Solutions for NP-Hard Problems*. Princeton, Feb. 23–24, 2000.
[60] Papadimitriou C H, Yannakakis M. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 1991, 43: 425–440.
[61] Impagliazzo R, Paturi R. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 2001, 63: 512–530.
[62] Abrahamson K, Downey R, Fellows M. Fixed-parameter tractability and completeness IV: On completeness of $W[P]$ and PSPACE analogs. *Ann. Pure Appl. Logic*, 1995, 73: 235–276.
[63] Downey R, Estivill-Castro V, Fellows M, Prieto-Rodriguez E, Rosamond F. Cutting up is hard to do: The parameterized complexity of k-cut and related problems. *Electronic Notes in Theoretical Computer Science*, 2003, 78: 205–218.
[64] Bodlaender H, Downey R, Fellows M, Hallett M, Wareham H. Parameterized complexity analysis in computational biology. *Computer Applications in Biosciences*, 1995, 11: 49–57.
[65] Cesati M. Compendium of parameterized problems (2004 version). Department of Computer Science, Systems, and Industrial Engineering, University of Rome "Tor Vergata", Italy.

http//bravo.ce.uniroma2.it/home/cesati/research/compendium.ps.

[66] Deng X, Li G, Li Z, Ma B, Wang L. Genetic design of drugs without side-effects. *SIAM J. Comput.*, 2003, 32: 1073–1090.

[67] Gramm J, Guo J, Niedermeier R. On exact and approximation algorithms for distinguishing substring selection. *Lecture Notes in Computer Science 2751* (*FCT*'03), 2003, pp.195–209.

[68] Chen J, Huang X, Kanj I A, Xia G. Linear FPT-reductions and computational lower bounds. In *Proc. 36th ACM Symp. Theory of Computing* (STOC 2004), 2004, pp.212–221.

[69] Goldschmidt O, Hochbaum D. Polynomial algorithm for the k-cut problem. In *Proc. 29th Ann. Symp. Foundations of Computer Science* (*FOCS*'88), 1988, pp.444–451.

[70] Bodlaender H, Fellows M, Hallett M. Beyond NP-completeness for problems of bounded width: Hardness for the W-hierarchy. In *Proc. 26th Ann. ACM Symp. Theory of Computing* (STOC'94), 1994, pp.449–458.

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

[72] Cai L, Juedes D. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 2003, 67: 789–807.

[73] Cai L, Chen J. On fixed parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 1997, 54: 465–474.

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

[75] Paz A, Moran S. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 1981, 15: 251–277.

[76] Woeginger G. When does a dynamic programming formulation guarantee the existence of an FPTAS? In *Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms* (SODA'99), 2001, pp.820–829.

[77] Kolaitis P, Thakur M. Approximation properties of NP minimization classes. *Journal of Computer and System Sciences*, 1995, 50: 391–411.

[78] Arora S, Lund C, Motwani R, Sudan M, Szegedy M. Proof verification and hardness of approximation problems. *Journal of the ACM*, 1998, 45: 501–555.

[79] Downey R. Parameterized complexity for the skeptic. In *Proc. 18th IEEE Conference on Computational Complexity* (*CCC*'03), 2003, pp.147–169.

[80] Cesati M, Trevisan L. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 1997, 64: 165–171.

[81] Khanna S, Motwani R. Towards a syntactic characterization of PTAS. In *Proc. 28th Ann. ACM Symp. Theory of Computing* (*STOC*'96), 1996, pp.329–337.

[82] Cai L, Fellows M, Juedes D, Rosamond F. Efficient polynomial-time approximation schemes for problems on planar structures: Upper and lower bounds. *Manuscript*, 2001.

[83] Huang X. *Parameterized complexity and polynomial-time approximation schemes* [Dissertation]. Department of Computer Science, Texas A&M University, December, 2004.

Jian-Er Chen got his B.S. degree in computer science in 1982 from Central South University, China, and his Ph.D. degree in computer science in 1987 from Courant Institute, New York University, USA, where he was awarded the Janet Fabri Award for the best Ph.D. dissertation. After graduation from NYU, he went to the Department of Mathematics at Columbia University, USA, where he received the Ph.D. degree in mathematics in 1990. Since then, he has been with the Department of Computer Science at Texas A&M University, USA, where he is a professor. Currently, he is a ChangJiang Scholar Professor at Central South University, China. His research interests include theoretical computer science, bioinformatics, computer networks, and computer graphics. He has published over 120 journal and conference papers in these areas, and received numerous awards, including the Research Initiation Award in 1991 from US National Science Foundation, TEES Select Young Faculty Award in 1993 and Distinguished Faculty Achievement Award in 1998 from Texas A&M University, Oversea Distinguished Young Scholars Award in 2000 from the National Natural Science Foundation of China, and Natural Science Award (first class) in 2003 from MOE, China.