Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 180–193
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Polynomial time approximation schemes and parameterized complexity

Jianer Chen[a,b,1], Xiuzhen Huang[c,2], Iyad A. Kanj[d,3], Ge Xia[e,4]

[a]*Department of Computer Science, Texas A&M University, College Station, TX 77843, USA*
[b]*College of Information Science and Engineering, Central South University, Changsha 410083, PR China*
[c]*Department of Computer Science, Arkansas State University, State University, AR 72467, USA*
[d]*School of CTI, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, USA*
[e]*Department of Computer Science, Lafayette College, Easton, PA 18042, USA*

Received 13 September 2004; received in revised form 4 September 2005; accepted 20 April 2006
Available online 3 July 2006

Abstract

In this paper, we study the relationship between the approximability and the parameterized complexity of NP optimization problems. We introduce a notion of *polynomial fixed-parameter tractability* and prove that, under a very general constraint, an NP optimization problem has a fully polynomial time approximation scheme *if and only if* the problem is polynomial fixed-parameter tractable. By enforcing a constraint of planarity on the *W*-hierarchy studied in parameterized complexity theory, we obtain a class of NP optimization problems, the *planar W-hierarchy*, and prove that all problems in this class have efficient polynomial time approximation schemes (EPTAS). The planar *W*-hierarchy seems to contain most of the known EPTAS problems, and is significantly different from the class introduced by Khanna and Motwani in their efforts in characterizing optimization problems with polynomial time approximation schemes.

Keywords: Approximation algorithm; Polynomial time approximation scheme; Parameterized complexity; *W*-hierarchy

1. Introduction

According to the NP-completeness theory [16], many optimization problems of theoretical interest and practical importance are NP-hard, thus cannot be solved optimally in polynomial time unless P=NP. Many approaches have been proposed to cope with the NP-hardness of such problems. The most celebrated among these approaches is polynomial time approximation, which involves compromising an optimal solution for a "good" solution that is computable in polynomial time.

[1] Supported in part by US NSF Grants CCR-0311590 and CCF-4030683, and by China NNSF Grants no. 60373083 and no. 60433020.
[2] Supported in part by US NSF Grant CCR-0000206.
[3] Supported in part by DePaul University Competitive Research Grant.
[4] Supported in part by US NSF Grant CCF-4030683.
E-mail addresses: chen@cs.tamu.edu (J. Chen), xzhuang@csm.astate.edu (X. Huang), ikanj@cs.depaul.edu (I.A. Kanj), gexia@cs.lafayette.edu (G. Xia).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.04.040

A notable class of NP-hard optimization problems has *fully polynomial time approximation schemes* (FPTAS), which are efficient approximation algorithms whose approximation ratio is bounded by $1+\varepsilon$ and whose running time is bounded by a polynomial in both input size and $1/\varepsilon$, where the relative error bound ε can be any positive real number. Examples of FPTAS problems include the well-known KNAPSACK problem and the MAKESPAN problem on a fixed number of processors [19]. A more general class of NP-hard optimization problems admits *polynomial time approximation schemes* (PTAS), which have polynomial time approximation algorithms of approximation ratio $1+\varepsilon$ for each fixed relative error bound $\varepsilon > 0$. A large number of NP-hard optimization problems, including, based on the recent research, the well-known EUCLIDEAN TRAVELING SALESMAN problem [2] and GENERAL MULTIPROCESSOR JOB SCHEDULING problem [11], belong to the class PTAS [19].

Contrary to the efficiency of FPTAS algorithms, the running time of a general PTAS algorithm of approximation ratio $1+\varepsilon$ can be of the form $O(n^{t(\varepsilon)})$, where n is the input size and $t(\varepsilon)$ is a function of ε that can be very large even for moderate values of ε. Downey [12] (see also [14]) examined many recently developed PTAS algorithms for NP-hard optimization problems, and discovered that for the relative error bound value of $\varepsilon = 20\%$, most of these PTAS algorithms have $t(\varepsilon) > 10^6$, i.e., the running time of these PTAS algorithms exceeds the order of $n^{100\,000}$! Obviously, these PTAS algorithms are not practically feasible.

Observing this fact, recent research has proposed to further refine the class PTAS. We say that an optimization problem Q has an *efficient polynomial time approximation scheme* (EPTAS) if for any $\varepsilon > 0$, there is an approximation algorithm of ratio $1+\varepsilon$ for Q whose running time is bounded by a polynomial of the input size whose degree is independent of ε. In particular, all FPTAS problems belong to the class EPTAS.

Clearly, EPTAS algorithms are superior to PTAS algorithms of running time of the form $O(n^{t(\varepsilon)})$ in the efficiency of the algorithms. In fact, many PTAS algorithms developed for NP-hard optimization problems are actually EPTAS algorithms. Moreover, there are a number of well-known NP-hard optimization problems, such as the EUCLIDEAN TRAVELING SALESMAN problem [2], the GENERAL MULTIPROCESSOR JOB SCHEDULING problem [11], and the MAKESPAN problem on unbounded number of processors [19], for which early developed PTAS algorithms had running time of form $O(n^{t(\varepsilon)})$, but later were improved to EPTAS algorithms. On the other hand, Cai et al. [7] recently studied the syntactic characterizations of PTAS problems proposed by Khanna and Motwani [21], and showed strong evidence that there are PTAS problems that have no EPTAS.

Therefore, it is interesting to investigate the characterization of EPTAS problems. The current paper attempts a study of EPTAS problems in terms of their parameterized complexity. Parameterized complexity theory [13] is a recently proposed new approach dealing with NP optimization problems, which studies the computational complexity of optimization problems in terms of both instance size and a properly selected parameter. The class FPT of *fixed-parameter tractable problems* has been introduced to characterize such parameterized problems that become feasible for small parameter values. On the other hand, a hierarchy, the *W-hierarchy* $\cup_{i \geqslant 1} W[i]$, has been introduced to capture the fixed-parameter intractability of optimization problems. Parameterized complexity theory has drawn considerable attention recently because of its applications in developing practical algorithms and in deriving computational lower bounds for NP optimization problems (see the surveys [12,14] for recent progress in the area).

We start by identifying a subclass, *PFPT*, of the class FPT, and prove that under a very general condition (the *scalability* condition, see Section 3 for a formal definition), a problem has FPTAS *if and only if* it is in PFPT. This provides the characterization for a very large subclass of the FPTAS problems in terms of parameterized complexity. This result seems to have advantages over the previous studies for the class FPTAS. Compared to Paz and Moran's characterization of the class FPTAS based on certain polynomial time computable functions [24] (see also [4]), our characterization seems easier to verify: the scalability condition seems to be satisfied by most NP optimization problems. Compared to Woeginger's recent study [26] on a subclass of the FPTAS problems based on a dynamic programming formulation, our characterization seems more general and to include more FPTAS problems.

We then study the characterization of the class EPTAS. We enforce a constraint of planarity on the *W*-hierarchy in parameterized complexity theory, and introduce the syntactic classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT (this approach is similar to that of Khanna and Motwani [21] in their efforts in characterizing the class PTAS). These syntactic classes capture many NP optimization problems in the class EPTAS, such as PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and PLANAR MAX-SAT. By extending Baker's techniques [5] and techniques more recently developed in the study of parameterized algorithms [1,15], we prove that all problems in these syntactic

classes belong to the class EPTAS. These syntactic classes seem to form the core for a significant class of EPTAS problems. Finally, we point out that our syntactic classes are significantly different from the PTAS syntactic classes introduced by Khanna and Motwani [21]: our syntactic classes contain only EPTAS problems while the syntactic classes in [21] seem to include PTAS problems that are not in EPTAS [7], while on the other hand, our syntactic classes contain EPTAS problems that cannot be characterized by the syntactic classes in [21].

Our results combined with a result by Cesati and Trevisan [8] show that all problems in our syntactic classes are fixed-parameter tractable. Moreover, a byproduct derived from an immediate result in our discussion shows that the PLANAR h-NORMALIZED WEIGHTED SATISFIABILITY problem is solvable in polynomial time, which answers an open problem posed by Downey and Fellows [13].

2. Preliminaries and further definitions

We review the necessary background in parameterized computation and in computational optimization. For more detailed discussions on these topics, the readers are referred to [3,13,16].

A *parameterized problem* Q is a subset of $\Omega^* \times N$, where Ω is a fixed alphabet and N is the set of all non-negative integers. Therefore, each instance of the parameterized problem Q is a pair (x, k), where the second component, i.e., the non-negative integer k, is called the *parameter*. We say that the parameterized problem Q is *fixed-parameter tractable* [13] if there is a (parameterized) algorithm that decides whether an input (x, k) is a member of Q in time $O(f(k)|x|^{O(1)})$, where $f(k)$ is any recursive function. Let FPT denote the class of all fixed parameter tractable problems.

Fixed-parameter intractability has been studied based on satisfiability problems on bounded-depth circuits. We review the related terminologies here. A (unbounded fan-in) *circuit* is a directed acyclic graph. The nodes of in-degree 0 are called *inputs*, and are labeled either by a *positive literal* x_i or by a *negative literal* $\overline{x}_i$. The nodes of in-degree larger than 0 are called *gates* and are labeled with a Boolean operator AND or OR. A special gate of out-degree 0 is designated as the *output* node. A circuit represents a Boolean function in a natural way. Without loss of generality, we can assume that circuits are of a special leveled form where all inputs are in level 0, and all AND and OR gates are organized into alternating levels with edges only going from a level to the next level [9]. The *depth* of a node v is d if v is at the dth level. The *depth* of a circuit is d if its output node has depth d. A circuit is a *Π_h-circuit* if it has depth h and its output node is an AND gate. We say that an input vector x of a circuit α *satisfies* a gate g in α if x makes the gate g have value 1, and that x *satisfies the circuit* α if x satisfies the output gate of α. The *weight* of an input vector x is the number of 1's in x. For a fixed integer $h \geqslant 2$, define the following parameterized problem:

$$\text{WCS}(h) = \{(\alpha, k) \mid \text{the } \Pi_h\text{-circuit } \alpha \text{ is satisfied by an input vector of weight } k\}.$$

The parameterized intractability classes, the *W-hierarchy* $\sum_{h \geqslant 1} W[h]$, are defined based on the problems WCS(h) via the *fpt-reduction* (see [13] for the definition for the fpt-reduction): a parameterized problem Q is in the class $W[h]$ if it is fpt-reducible to the WCS(h) problem.[5] A parameterized problem Q is $W[h]$-*hard* if every parameterized problem in $W[h]$ is fpt-reducible to Q, and is $W[h]$-*complete* if in addition Q itself is also in $W[h]$. It is widely believed that no $W[1]$-hard problem is fixed-parameter tractable [13].

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 $S_Q(x) = \{y : |y| \leqslant p(|x|) \& \pi(x, y)\}$.
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 $opt_Q = \min$. We will denote by $opt_Q(x)$ the value $opt_Q\{f_Q(x, z) \mid z \in S_Q(x)\}$.

Note that since the length of a solution y to an instance x in Q is bounded by a polynomial of $|x|$, and the objective function f_Q is computable in polynomial time, the values of the objective function f_Q, in particular the value $opt_Q(x)$, are bounded by $2^{q(|x|)}$ for a fixed polynomial q.

[5] The definition of the class $W[1]$ is different and somehow more technical. Since it is not directly related to our discussion, we refer the readers to [13] for the formal definition.

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

An NP optimization problem Q has a PTAS if there is an algorithm A_Q that takes a pair (x, ε) as input, where x is an instance of Q and $\varepsilon > 0$ is a real number, and returns a feasible solution y for x such that the approximation ratio of the solution y is bounded by $1 + \varepsilon$, and for each fixed $\varepsilon > 0$, the running time of the algorithm A_Q is bounded by a polynomial of $|x|$.[6] Finally, an NP optimization problem Q has an FPTAS if it has a PTAS A_Q such that the running time of A_Q is bounded by a polynomial of $|x|$ and $1/\varepsilon$.

Observe that the time complexity of a PTAS algorithm may be of the form $O(2^{1/\varepsilon}|x|^2)$ or of the form $O(|x|^{1/\varepsilon})$. Obviously, the latter type of computations with small ε values will turn out to be practically infeasible. This leads to the following definition [8].

Definition. An NP optimization problem Q has an *EPTAS* if it admits a PTAS whose time complexity is bounded by $O(f(1/\varepsilon)|x|^{O(1)})$, where f is a recursive function.

There is an essential difference between approximation algorithms and parameterized algorithms: 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 a meaningful comparison between approximability and parameterized complexity, we introduce a formal procedure that parameterizes an optimization problem and extend the definition of parameterized algorithms accordingly (see [6] for a similar treatment).

Definition. Let $Q = (I_Q, S_Q, f_Q, opt_Q)$ be an NP optimization problem.

- If $opt_Q = \max$, then the *parameterized version of* Q is $Q_{\geqslant} = \{(x, k) \mid x \in I_Q \& opt_Q(x) \geqslant k\}$.
- If $opt_Q = \min$, then the *parameterized version of* Q is $Q_{\leqslant} = \{(x, k) \mid x \in I_Q \& opt_Q(x) \leqslant k\}$.

A parameterized algorithm A_Q *solves the parameterized version of* Q if

- in case $opt_Q = \max$, on an input $(x, k) \in Q_{\geqslant}$, A_Q returns "yes" with a solution y in $S_Q(x)$ such that $f_Q(x, y) \geqslant k$, and on any input not in $Q_{\geqslant}$, A_Q simply returns "no";
- in case $opt_Q = \min$, on an input $(x, k) \in Q_{\leqslant}$, A_Q returns "yes" with a solution y in $S_Q(x)$ such that $f_Q(x, y) \leqslant k$, and on any input not in $Q_{\leqslant}$, A_Q simply returns "no".

The above definition allows us to consider the parameterized complexity of an NP optimization problem, which is the parameterized complexity of the parameterized version of the problem.

3. FPTAS and polynomial-FPT

Recall that a fixed-parameter tractable problem has an algorithm of running time of the form $f(k)n^{O(1)}$, where f is an arbitrary recursive function. By enforcing a further constraint on the function $f(k)$, we introduce the following subclass of the class FPT.

Definition. An NP optimization problem Q is *polynomial fixed-parameter tractable* (PFPT) if its parameterized version is solvable in time $O(|x|^{O(1)}k^{O(1)})$.

[6] There is an alternative definition for PTAS in which each $\varepsilon > 0$ may correspond to a different approximation algorithm A_ε for Q [16]. The definition we adopt here may be called the *uniform PTAS*, by which a single approximation algorithm takes care of all values of ε. Note that most PTAS developed in the literature are uniform PTAS.

Note that polynomial fixed-parameter tractability does not necessarily imply polynomial time computability: an NP optimization problem Q may have its optimal value $opt_Q(x)$ much larger than the instance size $|x|$. In consequence, the parameterized version of the problem Q may have its parameter values k larger than any polynomial of its instance size $|x|$.

The class of PFPT contains many important problems. For example, Cai and Chen [6] proved that every FPTAS problem is fixed-parameterized tractable. A more careful examination of their proof shows that in fact what they proved was that every FPTAS problem is in the class PFPT. In particular, a large variety of classical scheduling problems are in the class FPTAS [20,25], thus belong to the class PFPT.

In this section, we prove that the condition PFPT actually coincides with the condition FPTAS when an NP optimization problem is "scalable" in the following sense.

Definition. An optimization problem $Q = (I_Q, S_Q, f_Q, opt_Q)$ is *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 $|f_Q(x_d, y_d) - f_Q(x, y)/d| \leqslant q(|x|)$.

We give some explanation on the scalability property for NP optimization problems. For many NP optimization problems Q, an instance x consists of a set of weighted elements (plus certain logic structures on the elements). In this case, the element weights can be "scaled" (e.g., by dividing the element weights by an integer d), resulting in a new instance x' of Q. This operation of "scaling element weights" is formulated by the function g_1 given in the above definition in a more general form. The scalability property of Q requires that there be an association between the solution sets $S_Q(x)$ and $S_Q(x')$ such that if we also scale the solution value for a solution y in $S_Q(x)$ by the integer d, then the difference between the scaled solution value $f_Q(x, y)/d$ and the solution value $f_Q(x', y')$ for the corresponding solution y' for the scaled instance x' is bounded by a polynomial of $|x|$. In particular, if the solutions of the instance x consist of subsets of elements in x (such as weighted VERTEX COVER and DOMINATING SET), or of element ordering (such as TRAVELLING SALESMAN and various scheduling problems), then the scalability property is satisfied. Moreover, if an NP optimization problem Q has its optimal value $opt_Q(x)$ 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 $g_2(x_d, y_d) = y_d$.

To be more concrete, we pick $Q =$ MAKESPAN as an example to illustrate how an NP optimization problem can be scaled. An instance x of MAKESPAN consists of n jobs of integral processing times $t_1, t_2, \ldots, t_n$, respectively (we will refer to the jth job by t_j), and an integer m, the number of identical processors, and asks to construct a scheduling of the jobs on the m processors so that the completion time (i.e., the *makespan*) is minimized. For a given instance $x = (t_1, t_2, \ldots, t_n; m)$ of MAKESPAN and a given integer $d \geqslant 0$, we define

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

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

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

On the other hand,

$$f_Q(x_d, y_d) = \max_i \left\{ \sum_{t'_j \in T'_i} t'_j \right\} = \max_i \left\{ \sum_{t_j \in T_i} \lceil t_j/d \rceil \right\}$$
$$\leqslant \max_i \left\{ \sum_{t_j \in T_i} (t_j/d + 1) \right\} \leqslant \max_i \left\{ \sum_{t_j \in T_i} t_j \right\} /d + n = f_Q(x, y)/d + n \quad (2)$$

(note that the total number of jobs in each subset T_i is bounded by n). Combining (1) and (2), we get $|f_Q(x_d, y_d) - f_Q(x, y)/d| \leqslant n$. Similarly, it can be verified that the instances x and x_d satisfy $|opt_Q(x_d) - opt_Q(x)/d| \leqslant n$. In conclusion, the MAKESPAN problem is scalable.

Theorem 3.1. *Let $Q = (I_Q, S_Q, f_Q, opt_Q)$ be a scalable NP optimization problem. Then Q has an FPTAS if and only if Q is in PFPT.*

Proof. One direction of the theorem was implicitly proved in [6]. Suppose that Q has an FPTAS A_Q, which is an algorithm such that on any instance x of Q and any given $\varepsilon > 0$, the algorithm A_Q constructs a solution of ratio bounded by $1 + \varepsilon$ for x, in time $p(1/\varepsilon, |x|)$, where $p(1/\varepsilon, |x|)$ is a polynomial of $1/\varepsilon$ and $|x|$. Cai and Chen proved ([6, Theorem 3.2]) that then the parameterized version of Q can be solved in time $O(p(2k, |x|))$. In consequence, the problem Q is in PFPT.

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

1. let $x_1 = g_1(x, 1)$; if $(x_1, 3q(n)/\varepsilon)$ is not in $Q_{\geqslant}$, then try all instances $(x, 1), (x, 2), \ldots, (x, 3q(n)/\varepsilon + q(n))$ to construct an optimal solution for x; STOP;
2. use binary search on d to find an integer $d \geqslant 1$ such that $(x_d, 3q(n)/\varepsilon)$ is in $Q_{\geqslant}$, but $(x_{d+1}, 3q(n)/\varepsilon)$ is not in $Q_{\geqslant}$, where $x_d = g_1(x, d)$ and $x_{d+1} = g_1(x, d + 1)$;
3. construct an optimal solution y_d for the instance x_d;
4. let $y_0 = g_2(x_d, y_d)$; output y_0 as a solution for x.

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

If $(x_1, 3q(n)/\varepsilon)$ is in $Q_{\geqslant}$, then we execute step 2 of the algorithm. First, we need to show that there must be an integer $d \geqslant 1$ such that $(x_d, 3q(n)/\varepsilon)$ is in $Q_{\geqslant}$ but $(x_{d+1}, 3q(n)/\varepsilon)$ is not in $Q_{\geqslant}$. We already know that $(x_d, 3q(n)/\varepsilon)$ is in $Q_{\geqslant}$ for $d = 1$. Thus, we only need to show that there must be a d such that $(x_d, 3q(n)/\varepsilon)$ is not in $Q_{\geqslant}$. Since Q is an NP optimization problem, we have $opt_Q(x) < 2^{r(n)}$, where $r(n)$ is a polynomial in n. Therefore, if we let $d = 2^{r(n)}$, then from the scalability of the problem Q, we have $|opt_Q(x_d) - opt_Q(x)/d| \leqslant q(n)$, which gives immediately $opt_Q(x_d) < 1 + q(n) \leqslant 3q(n)/\varepsilon$ (here we assume without loss of generality that $q(n) \geqslant 1$ and $0 < \varepsilon < 1$). Thus, the integer d in step 2 of the algorithm must exist and $d \leqslant 2^{r(n)}$. Since we use binary search on d, the total number of instances $(x_d, 3q(n)/\varepsilon)$ we check in step 2 is bounded by $r(n)$. By our assumption, each instance $(x_d, 3q(n)/\varepsilon)$ of $Q_{\geqslant}$ can be tested in time $p(3q(n)/\varepsilon, q(n))$ (note that $|x_d| \leqslant q(n)$). Therefore, the running time of step 2 of the algorithm is also bounded by a polynomial of n and $1/\varepsilon$.

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

$$|opt_Q(x_d) - opt_Q(x)/d| \leqslant q(n) \quad \text{and} \quad |opt_Q(x_{d+1}) - opt_Q(x)/(d+1)| \leqslant q(n).$$

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

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

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

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

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

$$|opt_Q(x_d) - f_Q(x, y_0)/d| = |f_Q(x_d, y_d) - f_Q(x, y_0)/d| \leqslant q(n). \tag{3}$$

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

$$f_Q(x, y_0) \geqslant 3d \cdot q(n)/\varepsilon - d \cdot q(n) = q(n)(3d/\varepsilon - d) \geqslant 2dq(n)/\varepsilon. \tag{4}$$

Now from (3) and the inequality $|opt_Q(x_d) - opt_Q(x)/d| \leqslant q(n)$, we get

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

Thus $opt_Q(x) - f_Q(x, y_0) \leqslant 2dq(n)$. This gives us

$$opt_Q(x)/f_Q(x, y_0) \leqslant 1 + 2dq(n)/f_Q(x, y_0) \leqslant 1 + \varepsilon.$$

The last inequality is from (4). In conclusion, the approximation ratio of the solution y_0 for the instance x is bounded by $1 + \varepsilon$.

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

We make a few remarks on Theorem 3.1. Since the first group of publications on FPTAS for NP optimization problems [20,25], there has been a line of research trying to characterize problems in FPTAS [4,24,26]. Most of the early work in this direction [4,24] characterizes the class FPTAS in terms of certain polynomial time computable functions. These characterizations do not provide any clue on how to detect the existence of such functions, or on how to develop FPTAS for the problems (the interested readers are referred to [24], Theorem 4.20, for a more detailed discussion on this line of research). More recently, Woeginger [26], in an effort to overcome this difficulty, considered a class of optimization problems that can be formulated via dynamic programming of certain structures. He showed that as long as the cost and transition functions of such problems satisfy certain arithmetical and structural conditions, the problems have FPTAS.

Theorem 3.1 follows the same line of research as [26] but seems to have the following advantages when compared with previous works. First, as we have shown for the MAKESPAN problem, the scalability property of an NP optimization problem is satisfied in most cases and, in general, can be checked in a straightforward manner. Thus, in most cases, the existence of FPTAS for an NP optimization problem is reduced to the development of an PFPT algorithm for the problem. Moreover, the proof of Theorem 3.1 describes in detail how an PFPT algorithm is converted into an FPTAS algorithm. On the other hand, Theorem 3.1 seems to cover more FPTAS problems than Woeginger's formulation [26]: intuitively, and generally, a dynamic programming formulation for an NP optimization problem directly implies an PFPT algorithm for the problem.

4. Planar *W*-hierarchy and EPTAS

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

We note that a significant amount of research has been done on studying the approximation properties in terms of their syntactic descriptions. For instance, Papadimitriou and Yannakakis [23] introduced the syntactic classes MAXNP and MAXSNP of optimization problems, which, via proper approximation ratio preserving reductions, turn out to be exactly the class of NP optimization problems that can be approximated in polynomial time with constant approximation ratios [22]. Khanna and Motwani [21] proposed the syntactic classes MPSAT, TMAX, and TMIN by enforcing a planar structure on first order Boolean formulas of depth 3, and showed that most known PTAS problems are expressible by these classes.

In a parallel approach to that of Khanna and Motwani [21], we study the approximation class EPTAS by enforcing a planar structure on the *W*-hierarchy in parameterized complexity. A Π_h-circuit is a Π_h^+*-circuit* if all of its inputs are labeled by positive literals, and is a Π_h^-*-circuit* if all of its inputs are labeled by negative literals. A Π_h-circuit α is *planar* if α becomes a planar graph after removing the output gate in α.

Definition. We define the following syntactic optimization classes:

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

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

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

We make a few remarks on the above definitions. The classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT are optimization versions, with a planarity constraint, of the problem WCS(h), which is the representative complete problem for the hth level $W[h]$ of the *W*-hierarchy in parameterized complexity theory. Strictly speaking, PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR$W[h]$-SAT are optimization *problems*. We call them optimization *classes* because they characterize a large number of optimization problems (sometimes probably via a proper reduction).[7] The class PLANAR $W[h]$-SAT captures the optimization problems where the objective is to construct a solution that satisfies the maximum number of constraints. In particular, the problem PLANAR MAXSAT formulated by Khanna and Motwani [21] belongs to the class PLANAR $W[2]$-SAT. The classes PLANAR MIN-$W[h]$ and PLANAR MAX-$W[h]$ capture the optimization problems where the objective is to construct an optimal (minimum or maximum) solution that satisfies all the constraints. Most optimization problems on planar graphs belong to the classes PLANAR MIN-$W[h]$ or PLANAR MAX-$W[h]$. For example, for an instance G of the MINIMUM VERTEX COVER problem on planar graphs, we can convert G into a planar Π_2^+-circuit α_G by making each vertex v in G an input of α_G and replacing each edge $[v, w]$ in G by an OR gate with the two inputs v and w, which is connected to the unique output AND gate of the circuit α_G. It is easy to see that the minimum vertex covers of the graph G correspond to the minimum weight assignments that satisfy the circuit α_G, and vice versa.

In the rest of this section, we show that all optimization problems in our syntactic classes have EPTAS. EPTAS algorithms for these problems are developed based on methods similar to that in [5]. We provide the details below, emphasizing on the differences. Moreover, since the algorithms for the three classes are similar, we will concentrate on PLANAR MIN-$W[h]$, and give brief explanations on how the algorithms can be modified to apply to PLANAR MAX-$W[h]$ and PLANAR $W[h]$-SAT.

Let G be a planar graph (not necessarily connected) and $\pi(G)$ be a planar embedding of G. A vertex v is in *layer-1* in $\pi(G)$ if v is on the boundary of the unbounded region of $\pi(G)$. We define G_1 to be the subgraph of G induced by all layer-1 vertices. Inductively, a vertex v is in layer-i, $i > 1$, if v is on the boundary of the unbounded region of the

[7] This is similar to the approaches that have been adopted by Papadimitriou and Yannakakis in their study of the class SNP [23] and that by Khanna and Motwani in their study of the PTAS class [21].

embedding of the graph $G - (G_1 \cup \cdots \cup G_{i-1})$ induced by the embedding $\pi(G)$. Define G_i to be the subgraph of G induced by all layer-i vertices. The embedding $\pi(G)$ is *q-outerplanar* if it has at most q layers.

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

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

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

Lemma 4.1. *The problem* MIN(h, q)-SAT *can be solved in time* $O(81^q n)$.

Proof. The proof proceeds based on the techniques proposed by Baker [5]. 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 (the left and right boundaries are not necessarily disjoint). A *trivial slice* is simply an edge in G. Two slices S_1 and S_2 can be "merged" into a larger slice S if the right boundary of S_1 is identical to the left boundary of S_2. Baker [5] presented a linear-time algorithm to show how a q-outerplanar graph G is recursively decomposed into trivial slices and how the slices, starting from trivial slices, are recursively merged to reconstruct the original graph G.

Each vertex in the graph G is either an input variable or a gate in the original circuit. Therefore, in order to solve the problem MIN (h, q)-SAT, for each slice S, we need to assign proper values to the input variables in S and determine the corresponding values for gates in S. For this, starting from the trival slices, we recursively examine every possible value assignment to the boundary vertices of each slice S, and record the minimum weight assignment to the input variables in S that induces the boundary vertex assignment and satisfies all depth-$(h-1)$ gates in S that are in the middle $q - 2h$ layers in $\pi(G)$ (as long as such an assignment exists). Note that the value of a boundary vertex v in S may be determined by an input of v that is not in S. For example, an AND gate v in S may have value 0 because an input of v not in S has value 0 (and all inputs of v in S have value 1). To indicate this special case, we introduce the values $\tilde{0}$ and $\tilde{1}$ as follows: an AND gate in the slice S has value $\tilde{0}$ if all of its inputs in S have value 1 but an input of it not in S has value 0; and an OR gate in S has value $\tilde{1}$ if all of its inputs in S have value 0 but an input of it not in S has value 1. Note that the gate values $\tilde{0}$ and $\tilde{1}$ cannot be verified until the slice S is merged with other slices. Therefore, when we work on the slice S, each boundary vertex v in S may have one of the following values:

- If v is an input variable, then v has value either 0 or 1.
- If v is an OR gate, then v has one of the following values:
  (1) value 0 (assuming all inputs of v, in particular all those in S, have value either 0 or $\tilde{0}$);
  (2) value1 (assuming an input of v in S has value 1); or
  (3) value $\tilde{1}$ (assuming no input of v in S has value 1 but an input of v not in S has value 1).
- If v is an AND gate, then v has one of the following values:
  (1) value 1 (assuming all inputs of v, in particular all those in S, have value either 1 or $\tilde{1}$);
  (2) value 0 (assuming an input of v in S has value 0); or
  (3) value $\tilde{0}$ (assuming no input of v in S has value 0 but an input of v not in S has value 0).

We point out that since an input to an OR gate is either an input variable or an AND gate, no input of an OR gate can have value $\tilde{1}$. Similarly, no input of an AND gate can have value $\tilde{0}$.

We call a possible value assignment to all vertices in a (left or right) boundary of a slice a "configuration" of the boundary. Each slice S with left boundary L and right boundary R is associated with a "table" T_S. For each configuration

f_L of L and each configuration f_R of R, the table T_S records a minimum weight assignment $A_{\min}(S, f_L, f_R)$ to the input variables in the slice S that realizes the configurations f_L and f_R on the boundaries L and R, and satisfies all depth-$(h-1)$ gates that are in S and belong to the middle $q-2h$ layers in $\pi(G)$. Since each vertex in L and R may have at most three different values and the total number of vertices in $L \cup R$ is bounded by $2q$, the table T_S has at most 3^{2q} items.

If S is a trivial slice, then the table T_S can be constructed by brute force. Now we illustrate how the value $A_{\min}(S, f_L, f_R)$ is computed for a larger slice S, where S is obtained from merging two smaller slices S_1 and S_2. Inductively, assume that the tables T_{S_1} and T_{S_2} have been constructed. Let the left and right boundaries of S_1 and S_2 be L_1 and R_1, and L_2 and R_2, respectively. By the construction described in [5], $R_1 = L_2$, and the left and right boundaries of the larger slice S are $L = L_1$ and $R = R_2$.

Let f_{R_1} and f_{L_2} be configurations of the boundaries R_1 and L_2, respectively, and let v be a vertex on $R_1 = L_2$. We say that the assignments of f_{R_1} and f_{L_2} on v are "consistent" if:

(C1) $f_{R_1}(v) = f_{L_2}(v) = 0$ or $f_{R_1}(v) = f_{L_2}(v) = 1$; or
(C2) v is an OR gate, $f_{R_1}(v) = f_{L_2}(v) = \tilde{1}$, and v has an input not in $S_1 \cup S_2$; or
(C3) v is an AND gate, $f_{R_1}(v) = f_{L_2}(v) = \tilde{0}$, and v has an input not in $S_1 \cup S_2$; or
(C4) v is an OR gate, and one of $f_{R_1}(v)$ and $f_{L_2}(v)$ is 1 and the other is $\tilde{1}$; or
(C5) v is an AND gate, and one of $f_{R_1}(v)$ and $f_{L_2}(v)$ is 0 and the other is $\tilde{0}$.

Note that if the vertex v is also a boundary vertex for the slice S after merging S_1 and S_2, then v also naturally gets a value: by the definitions, in cases (C1)–(C3), the vertex v will have value $f_{R_1}(v) = f_{L_2}(v)$, while in case (C4) v gets value 1 and in case (C5) v gets value 0.

Finally, we say that the configurations f_{R_1} and f_{L_2} are *consistent* if their value assignments to every vertex on $R_1 = L_2$ are consistent.

The key observation is that every assignment to the input variables in the slice S corresponds to an assignment to the input variables in the slice S_1 and an assignment to the input variables in the slice S_2. Therefore, the minimum weight assignment $A_{\min}(S, f_L, f_R)$ for the slice S can be derived from a minimum weight assignment $A_{\min}(S_1, f_L, f_{R_1})$ in T_{S_1} and a minimum weight assignment $A_{\min}(S_2, f_{L_2}, f_R)$ in T_{S_2} for some consistent configurations f_{R_1} and f_{L_2}. Since inductively the tables T_{S_1} and T_{S_2} are already available, we can enumerate all consistent configurations f_{R_1} and f_{L_2} and the corresponding minimum weight assignments $A_{\min}(S_1, f_L, f_{R_1})$ in T_{S_1} and $A_{\min}(S_2, f_{L_2}, f_R)$ in T_{S_2}, then derive the minimum weight assignment $A_{\min}(S, f_L, f_R)$. Since each boundary vertex v may have three possible values, and each (left or right) boundary has at most q vertices, for a fixed pair of configurations f_L and f_R of the boundaries L and R, there are at most $3^q \cdot 3^q$ possible pairs of configurations on the boundaries R_1 and L_2. Thus, the minimum weight assignment $A_{\min}(S, f_L, f_R)$ can be constructed in time $O(9^q)$. In consequence, the table T_S, which has a record for each pair of configurations f_L and f_R of the boundaries L and R, can be constructed in time $O(9^q \cdot 9^q) = O(81^q)$.

Using Baker's linear-time algorithm that recursively decomposes the q-outerplanar graph G into slices and reconstructs the graph G from its trivial slices by recursively merging slices, we conclude that in time $O(81^q n)$, we can construct a minimum weight assignment to the input variables in G that satisfies all depth-$(h-1)$ gates that are in the middle $q-2h$ layers in G, thus solving the MIN (h, q)-SAT problem. This completes the proof.[8] □

Downey and Fellows [13, p. 482] posed an open problem for the parameterized complexity of the following problem:

> PLANAR h-NORMALIZED WEIGHTED SATISFIABILITY: Given a Π_h-circuit α that is a planar graph in the strict sense (i.e., it is planar even without removing the output gate) and a parameter k, does α have a satisfying assignment of weight k?

Using the techniques presented in Lemma 4.1, we can solve this open problem completely.

[8] We remark that based on the approach of graph tree decomposition and more careful slice merging [1], the complexity of the algorithm described in Lemma 4.1 can be improved to $O(c^q n)$ for a constant c much smaller than 81. However, this will not affect our main results in this paper.

Theorem 4.2. *For each integer h, the* PLANAR h-NORMALIZED WEIGHTED SATISFIABILITY *problem is solvable in polynomial time.*

Proof. Fix a planar embedding $\pi(\alpha)$ of the circuit α. Suppose the output gate of α is contained in the ith layer L_i in $\pi(\alpha)$. Since the depth of α is bounded by h, every gate in α must be contained in one of the $2h+1$ layers $L_{i-h}, \ldots, L_i, \ldots, L_{i+h}$. In consequence, the embedding $\pi(\alpha)$ must be $(2h+1)$-outerplanar. Thus, similar to the proof of Lemma 4.1, we can construct a satisfying assignment of weight k to the circuit α based on the slice structure of $\pi(\alpha)$ (or report that no such an assignment exists). The only difference is that now for each boundary configuration (f_L, f_R) of a slice S, we should record *all* possible weights w, $0 \leqslant w \leqslant k$, such that there is a weight-w assignment to the input variables in the slice S that satisfies all the depth-$(h-1)$ gates in S and implements the boundary configuration (f_L, f_R). Now merging two slices should also consider combining all possible weights recorded in the two slices, which increases the time complexity by a factor of $O((2h+1)^2)$. Therefore, for a fixed integer h, this induces a linear-time algorithm (of running time $O(81^{2h+1}(2h+1)^2 n)$) for PLANAR h-NORMALIZED WEIGHTED SATISFIABILITY. □

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

Theorem 4.3. *For every $h \geqslant 1$,* PLANAR MIN-$W[h]$ *is a subclass of the class EPTAS.*

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

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

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

Let U_i be the ith chunk of G and $\pi(U_i)$ be the q-outerplanar embedding of U_i induced from the embedding $\pi(G)$. According to Lemma 4.1, in time $O(81^q n_i)$ we can construct a minimum weight assignment f_{d,U_i} to the input variables in U_i that satisfies all depth-$(h-1)$ gates in the middle $q-2h$ layers in $\pi(U_i)$, where n_i is the total number of vertices in U_i.

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

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

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

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

Now suppose that f_{opt} is a minimum weight assignment of weight $|f_{\text{opt}}|$ to the input variables in G_w that satisfies the circuit G_w. Let V_{opt} be the set of input variables which are assigned value 1 by f_{opt} (thus, $|f_{\text{opt}}| = |V_{\text{opt}}|$). Since the layer sets S_i, $0 \leqslant i \leqslant q/(2h) - 2$, are disjoint, one of the layer sets contains at most $|f_{\text{opt}}|/(q/(2h) - 1) \leqslant \varepsilon \cdot |f_{\text{opt}}|$ input variables in V_{opt}. Let this set be S_d, and let V^d_{opt} be the set of input variables in both V_{opt} and S_d, $|V^d_{\text{opt}}| \leqslant \varepsilon \cdot |f_{\text{opt}}|$. We consider the chunk decomposition D_d. Let f_d be the assignment to the input variables in G_w constructed by our algorithm based on the chunk decomposition D_d.

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

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

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

$$\sum_{i=0}^{p} |f_{\text{opt},U_i}| = |f_{\text{opt}}| + |V^d_{\text{opt}}| \leqslant |f_{\text{opt}}|(1+\varepsilon).$$

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

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

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

We briefly describe how Lemma 4.1 and Theorem 4.3 are modified to apply to the classes PLANAR MAX-$W[h]$ and PLANAR $W[h]$-SAT.

Given an instance G_w of a PLANAR MAX-$W[h]$ problem and a real number $\varepsilon > 0$, where G_w is a planar Π_h^--circuit with the output gate w, we let $q = h(\lceil 1/\varepsilon_0 \rceil + 1)$, where $\varepsilon_0 = \varepsilon/(1+\varepsilon)$, and construct a planar embedding $\pi(G)$ of the graph $G = G_w - w$. Now each chunk decomposition D_d partitions the graph G into *disjoint* chunks, each consists of q consecutive layers in $\pi(G)$. The first h layers and the last h layers in a chunk will be called the *boundary layers* of the chunk. Assign value 1 to input gates that are in boundary layers of the chunks. Since all input gates are labeled by negative literals, this assignment is equivalent to assigning value 0 to the corresponding input variables. According to this assignment, if a gate g_1 has an input from a gate g_2 such that g_1 and g_2 belong to two different chunks, then the gate g_2 must have value 1 since all input gates that can affect the gate g_2 are in boundary layers and hence have been assigned value 1.

With this initial assignment, now we work on each chunk U in D_d. Note that it is always possible to assign the remaining input gates in the chunk U to satisfy all depth-$(h-1)$ gates in U (e.g., assigning all remaining input gates in U value 1, or equivalently, assigning all remaining input variables in U value 0). Since the chunk U is given as its q-outerplanar embedding induced from $\pi(G)$, using the techniques similar to that of Lemma 4.1, we can construct a maximum weight assignment to the remaining input variables in U that satisfies all depth-$(h-1)$ gates in U. As shown in Lemma 4.1, such an assignment can be constructed in time $O(81^q n_U)$, where n_U is the number of vertices in U. Doing this for all chunks in the chunk decomposition D_d gives an assignment f_d to the input variables that satisfies all depth-$(h-1)$ gates in G_w, thus satisfying the circuit G_w. Now we apply this process to each possible chunk decomposition D_d, each gives an assignment f_d satisfying the circuit G_w. We pick the one, denoted by f_{apx}, with the

largest weight among all f_d's and output it as the approximation solution to the problem. The assignment f_{apx} can be constructed in time $O(81^q n^2) = O(81^{O(1/\varepsilon)} n^2)$.

Similar to the analysis given in Theorem 4.3, if we fix an optimal assignment f_{opt} to the circuit G_w, then there is a chunk decomposition D_d in which the number m_d of variables that are in the boundary layers of D_d and are assigned value 1 by the assignment f_{opt} is bounded by $\varepsilon_0|f_{\text{opt}}|$. Moreover, the weight of the assignment f_d constructed based on the chunk decomposition D_d is at least $|f_{\text{opt}}| - m_d$. Since the weight of the assignment f_{apx} is the largest among all f_d's, we conclude that the assignment f_{apx} has weight at least $|f_{\text{opt}}| - m_d$. In consequence, the ratio $|f_{\text{apx}}|/|f_{\text{opt}}|$ is at least $1 - \varepsilon_0$. Replacing ε_0 by $\varepsilon/(1+\varepsilon)$ gives the approximation ratio $|f_{\text{opt}}|/|f_{\text{apx}}| \leqslant 1 + \varepsilon$. This completes the proof that every problem in PLANAR MAX-$W[h]$ is in the class EPTAS.

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

Theorem 4.4. *For every $h \geqslant 1$,* PLANAR MAX-$W[h]$ *and* PLANAR $W[h]$-SAT *are subclasses of the class EPTAS.*

Cesati and Trevisan [8] proved that if an optimization problem is in the class EPTAS then its parameterized version is fixed-parameter tractable. Combining this with Theorem 4.3 and Theorem 4.4, we get the following:

Corollary 4.5. *For every positive integer h, the classes* PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, *and* PLANAR $W[h]$-SAT *are subclasses of FPT.*

Finally, we point out that our syntactic classes PLANAR MIN-$W[h]$, PLANAR MAX-$W[h]$, and PLANAR $W[h]$-SAT are significantly different from the classes TMIN, TMAX, and MPSAT proposed by Khanna and Motwani [21] in the following sense. First, Corollary 4.5 proves that all optimization problems in our syntactic classes are fixed-parameter tractable, while Cai et al. [7] recently proved that there are $W[1]$-hard problems in the syntactic classes introduced in [21], which therefore should not be contained in our syntactic classes unless an unlikely collapse in parameterized complexity theory occurs. On the other hand, our classes are not subclasses of that of Khanna and Motwani's: the classes TMIN, TMAX, and MPSAT are defined based on circuits of depth 3, while ours are defined based on circuits of any constant depth. According to the well-known research on constant depth circuits [18], the classes PLANAR $W[h]$-SAT, PLANAR MIN-$W[h]$, and PLANAR MAX-$W[h]$ for $h > 3$ *cannot* be expressed by the syntactic classes TMIN, TMAX, and MPSAT.

References

[1] J. Alber, H. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, Algorithmica 33 (2002) 461–493.

[2] S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, J. ACM 45 (1998) 753–782.

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

[4] G. Ausiello, A. Marchetti-Spaccamela, M. Protasi, Toward a unified approach for the classification of NP-complete optimization problems, Theoret. Comput. Sci. 12 (1980) 83–96.

[5] B. Baker, Approximation algorithms for NP-complete problems on planar graphs, J. ACM 41 (1994) 153–180.

[6] L. Cai, J. Chen, On fixed-parameter tractability and approximability of NP optimization problems, J. Comput. System Sci. 54 (1997) 465–474.

[7] L. Cai, M. Fellows, D. Juedes, F. Rosamond, On efficient polynomial-time approximation schemes for problems on planar structures, Preprint, 2002.

[8] M. Cesati, L. Trevisan, On the efficiency of polynomial time approximation schemes, Inform. Process. Lett. 64 (1997) 165–171.

[9] J. Chen, Characterizing parallel hierarchies by reducibilities, Inform. Process. Lett. 39 (1991) 303–307.

[11] J. Chen, A. Miranda, A polynomial time approximation scheme for general multiprocessor job scheduling, SIAM J. Comput. 31 (2001) 1–17.

[12] R. Downey, Parameterized complexity for the skeptic, in: Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC'03), 2003, pp. 132–153.

[13] R. Downey, M. Fellows, Parameterized Complexity, Springer, Berlin, 1999.

[14] M. Fellows, Parameterized Complexity: the Main Ideas and Some Research Frontiers, Lecture Notes in Computer Science, vol. 2223 (ISAAC'01), Springer, Berlin, 2001, pp. 291–307.

[15] M. Frick, M. Grohe, Deciding first-order properties of locally tree-decomposable structures, J. ACM 48 (2001) 1184–1206.

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

[18] J. Hastad, Computational Limitations for Small-Depth Circuits, The MIT Press, Cambridge, MA, 1986.
[19] D. Hochbaum, Approximation Algorithms for NP-hard Problems, PWS Publishing Company, Boston, MA, 1997.
[20] O. Ibarra, C. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, J. ACM 22 (1975) 463–468.
[21] S. Khanna, R. Motwani, Towards a syntactic characterization of PTAS, in: Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96), 1996, pp. 468–477.
[22] S. Khanna, R. Motwani, M. Sudan, U. Vazirani, On syntactic versus computational views of approximability, SIAM J. Comput. 28 (1998) 164–191.
[23] C. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, J. Comput. System Sci. 43 (1991) 425–440.
[24] A. Paz, S. Moran, Non deterministic polynomial optimization problems and their approximations, Theoret. Comput. Sci. 15 (1981) 251–277.
[25] S. Sahni, Algorithms for scheduling independent tasks, J. ACM 23 (1976) 116–127.
[26] G. Woeginger, When does a dynamic programming formulation guarantee the existence of an FPTAS?, in: Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'99), 2001, pp. 820–829.