On the Structure of Parameterized Problems in NP *
(Extended Abstract, to appear STACS '94)

Liming Cai[1] Jianer Chen[1] Rodney Downey[2] Michael Fellows[3]

July 7, 2010

Abstract

Fixed-parameter intractability of optimization problems in *NP* is studied based on computational models with limited nondeterminism. Strong evidence is provided that many *NP* optimization problems are not fixed-parameter tractable and that the fixed-parameter intractability hierarchy (the W-hierarchy) does not collapse.

1 Introduction

A theory of fixed-parameter tractability of *NP* optimization problems has been initialized recently by Downey and Fellows [**?**, **?**, **?**] with the aim of refining the class of *NP* optimization problems and of solving *NP* optimization problems in practice. They have observed that many *NP* optimization problems can be parameterized, while the complexity of these problems may vary very differently with respect to the parameter. For example, the problem of finding a size k vertex cover in a graph can be solved in time $O(n^c)$, where

*Cai is supported in part by Engineering Excellence Award from Texas A&M University; Chen is supported in part by NSF Grant CCR-9110824; Downey is supported in part by a grant from the Victoria University IGC, by the United States/New Zealand Cooperative Science Foundation under grant INT 90-20558, and by the Mathematical Sciences Institute at Cornell and Cornell University; and Fellows is supported in part by the National Science and Engineering Research Council of Canada, and by the United States NSF under grant MIP-8919312.

c is a constant independent of the parameter k (in fact, $c = 1$ by [**?**]); while the problem of finding a size k dominating set in a graph has the contrasting situation where essentially no better algorithm is known than the "trivial" algorithm of time $O(n^{k+1})$ that just exhaustively tries all possible solutions.

In order to capture the fixed-parameter intractability of *NP* optimization problems, Downey and Fellows have introduced a hierarchy (called the *W-hierarchy*)

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P]$$

Sitting at the bottom of the hierarchy is the class FPT of fixed-parameter tractable problems that can be solved for each fixed k in time $O(n^c)$ where c is a constant independent of the parameter k. Examples in the class FPT are *Vertex-Cover* and *Minimum-Genus* [**?**, **?**, **?**]. More than a dozen problems, including *Independent-Set*, have been shown to be complete for the class $W[1]$ [1], while *Dominating-Set* and a number of other problems are shown to be complete for the class $W[2]$. A typical complete problem for the "cap class" $W[P]$ of the hierarchy is the *Weighted Circuit Satisfiability* problem. it is conjectured that the W-hierarchy is infinite [**?**]. Therefore, the completeness of a problem at some level of the W-hierarchy indicates the computational difficulty of the problem with respect to the parameter. The completeness theory for parameterized computational complexity has been shown to have many applications in diverse problem domains including familiar graph-theoretical problems, VLSI layout, games, computational biology, cryptography, and computational learning [**?**, **?**, **?**, **?**].

Another motivation of the study of fixed-parameter intractability is due to its close connection to the approximability of *NP*-hard optimization problems. As shown by Cai and Chen [**?**], with a minor restriction, the approximability of an optimization problem implies the fixed-parameter tractability of the problem. Therefore, the completeness study of the W-hierarchy provides a useful tool for proving the non-approximability of problems: showing an optimization problem to be hard for some level in the W-hierarchy implies that the problem is not approximable unless the W-hierarchy collapses.

Therefore, it has wide-ranging practical and theoretical significance to show that the W-hierarchy does not collapse. However, it may seem a bit ambitious to derive a direct proof since any separation result for the W-hierarchy would imply $P \neq NP$. Thus, it may be instead more feasible

[1] The completeness in the W-hierarchy is based on the reduction called "*uniform reduction*" that will be described precisely in Section 2.

to show that the collapsing of the W-hierarchy implies unlikely results in classical complexity theory.

In the present paper we study the structural properties of the W-hierarchy based on computational models with limited nondeterminism [**?**]. By techniques of inverse function, we are able to show that a parameterized problem is fixed-parameter tractable if and only if it can be solved by a polynomial time algorithm that is allowed to guess a string of length $f(k)$, while a parameterized problem is in the class $W[P]$ if and only if it can be solved by a polynomial-time algorithm that is allowed to guess a string of length $g(k)\log n$, where $f(k)$ and $g(k)$ are arbitrary recursive functions independent of the input length n. These characterizations of the classes FPT and $W[P]$ indicates a clear difference between the two classes. Therefore, to find a vertex cover of size k in a graph, we only need to guess a string of length $f(k)$, no matter how large the graph is, where $f(k)$ is a fixed recursive function depending only on the parameter k; while to find a weight k satisfying assignment for a circuit C, we *must* be able to guess a string whose length is of the form $g(k)\log|C|$ for some recursive function $g(k)$ and increases with the size $|C|$ of the circuit C.

We also derive a characterization for each level of the W-hierarchy by computations with limited nondeterminism, and show a strong evidence supporting the conjecture that the W-hierarchy does not collapse. More specifically, for each level $W[t]$ of the W-hierarchy, we introduce a computation model with the ability of making limited nondeterminism and show that this model defines a subclass of $W[t]$ that is equivalent to the class $W[t]$ up to the uniform reduction. Then we prove that the defined subclass of $W[t]$ and the defined subclass of $W[t+1]$ are distinct for all $t \geq 1$.

Another interesting question is whether approximability and fixed-parameter tractability are equivalent for computational optimization problems. Roughly speaking, approximability implies fixed-parameter tractability [**?**]. However, there are problems, such as *Longest Path*, that are fixed-parameter tractable but not approximable [**?**, **?**]. Having observed this fact, we further refine the class FPT with the intention of specifying the approximability of the problems. We show that the class FPT can be further classified in term of the ability of guessing and the power of verifying. Strong evidence is given that these subclasses are distinct.

The paper is organized as follows. In Section 2 we introduce the necessary definitions and preliminaries. The characterizations of the classes FPT and $W[P]$ in terms of the computation models with limited nondeterminism are

given in Section 3. The structural properties of the W-hierarchy are discussed in Section 4. Section 5 discusses the refinement of the class FPT. Conclusions are given in Section 6.

2 Preliminaries

We first give a brief review on the fundamentals of the theory of fixed-parameter tractability. For detailed description, see [**?**, **?**].

Definition 2.1. *A parameterized problem L is a subset of $\Sigma^* \times N$, where Σ is a fixed alphabet. Therefore, each instance of the parameterized problem L is a pair $\langle x, k\rangle$, where the second component k will be called the parameter.*

The complexity of a parameterized problem can be specified in terms of the two components of its instances.

Definition 2.2. *A parameterized problem L is (strongly) fixed-parameter tractable if there is an algorithm to decide whether $\langle x, k\rangle$ is a member of L in time $f(k)|x|^c$, where $f(k)$ is a recursive function and c is a constant independent of the parameter k. Let FPT denote the class of fixed-parameter tractable problems.*

Definition 2.3. *Let L and L' be two parameterized problems. We say that L is (uniformly) reducible to L' if there is an algorithm M that transforms $\langle x, k\rangle$ into $\langle x', g(k)\rangle$ in time $f(k)|x|^c$, where f and g are recursive functions and c is a constant independent of k, such that $\langle x, k\rangle \in L$ if and only if $\langle x', g(k)\rangle \in L'$.*

It is easy to observe that if L is reducible to L' and L' is fixed-parameter tractable, then so is L.

To define the W-hierarchy, we need the following notations similar to those introduced by Boppana and Sipser [**?**]. We say a circuit C is a Π_t^h-circuit if C is of unbounded fan-in and of depth at most $t+1$ with an AND gate at the output and gates of fan-in at most h at the input level. Let t and h be two integers, we define a parameterized problem as follows:

$$WCS(t,h) = \{(C,k) \mid C \text{ is a } \Pi_t^h\text{-circuit and accepts a weight } k \text{ input vector}\}$$

Definition 2.4. *A parameterized problem L belongs to the class $W[t]$ if L is reducible to the parameterized problem $WCS(t,h)$ for some constant h.*

The class $W[P]$ is defined similarly as $W[t]$ with no restriction on the depth of the circuits. Formally,

Definition 2.5. *A parameterized problem L belongs to the class $W[P]$ if L is reducible to the following Weighted Circuit Satisfiability (WSC) problem:*

$$WCS = \{(C, k) \mid \textit{The circuit } C \textit{ accepts an input vector of weight } k\}$$

The above leads to an interesting hierarchy (called the *W-hierarchy*)

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P]$$

for which a wide variety of natural problems are now known to be complete or hard for various levels (under the uniform reduction) [**?**].

The structural properties of the W-hierarchy will be studied based on the following GC model (for "Guess-then-Check") introduced by Cai and Chen [**?**].

Definition 2.6. *Let $s(n, k)$ be a recursive function on two variables. A parameterized problem L is in $GC(s(n, k), P)$ if there is a deterministic algorithm M and a polynomial q such that for all $z = \langle x, k\rangle$, $z \in L$ if and only if $\exists y \in \{0, 1\}^*$, $|y| \leq s(|x|, k)$, and M accepts $z\#y$ in time $O(q(|z|))$.*

Intuitively, the first component $s(n, k)$ in the GC model specifies the length of the guessed string y, which is the amount of nondeterminism allowed to make in the computation, while the second component P (polynomial time computation) specifies the power of verifying of the computation. Note that the complexity the verifying algorithm M is measured by the size of z rather than the size of the input $z\#y$ to the algorithm M. Similarly, the GC model may also be defined by substituting the complexity class P by other complexity classes such as DL, NL, and R.

The GC model is quite robust. It can be shown that for many functions $s(n, k)$ and for many complexity classes $\mathcal{C}$, the class $GC(s(n, k), \mathcal{C})$ has natural complete languages [**?**]. We also point out that some restricted forms of the GC model have been studied recently in the literatures [**?**, **?**, **?**, **?**].

For each fixed integer r, let $R(x_1, \cdots, x_r)$ be the set of all recursive functions of r variables. Let $\mathcal{C}$ be an arbitrary complexity class, we will define

$$GC(R(n, k), \mathcal{C}) \qquad GC(R(k), \mathcal{C}) \qquad GC(R(k) \log n, \mathcal{C})$$

to be the following classes, respectively.

$$\bigcup_{f\in R(n,k)} GC(f(n,k),\mathcal{C}) \qquad \bigcup_{f\in R(k)} GC(f(k),\mathcal{C}) \qquad \bigcup_{f\in R(k)} GC(f(k)\log n,\mathcal{C})$$

Remarks:

1. Parameterized problems form a restricted subclass in the class of general problems. Therefore, we can talk about the complexity of a parameterized problem in terms of the instance size. For example, we say that a parameterized problem is in the class NP if there are a nondeterministic algorithm M and a polynomial r such that M can decide whether a given instance $z = \langle x, k\rangle$ is a member of L in time $r(|z|)$.

2. Since a parameterless problem can always be regarded as a parameterized problem with a dummy parameter, the model GC can also be used to define classes of general problems.

3 The characterizations of FPT and $W[P]$

We start with the following simple observation.

Theorem 3.1. *Every parameterized problem in NP is in the class* $GC(R(n,k),P)$.

Recall that a function $t(n)$ is *time-constructible* if there is a deterministic algorithm M of running time $O(t(n))$ such that given input 1^n, M gives as the output $1^{t(n)}$ [**?**]. We need to introduce the concept of log-space constructibility of functions.

Definition 3.2. *A function* $s(n)$ *is* log*-space constructible if there is a deterministic algorithm* M *that runs in space* $O(\log(s(n))$ *such that given input* 1^n, M *outputs* $s(n)$ *in binary form.*

Lemma 3.3. *For any recursive function* $r(n)$, *there is a* log*-space constructible non-decreasing function* $f(n) \geq n$ *such that* $f(n) \geq r(n)$ *for all* $n \geq 1$.

Let f be an unbounded non-decreasing function. The *inverse function* f^{-1} of f is defined as follows:

$$f^{-1}(m) = \max\{n \mid f(n) \leq m\}$$

It is easy to see that f^{-1} is well-defined, unbounded, and non-decreasing.

Lemma 3.4. *Let $f(n)$ be an unbounded non-decreasing function and let k be an arbitrary integer. Then $f(k) \leq n$ if and only if $f^{-1}(n) \geq k$.*

Lemma 3.5. *Let $f(n) \geq n$ be an unbounded, non-decreasing and* log*-space constructible function. Then the inversion function $f^{-1}(n)$ of f is computable by a deterministic algorithm whose space is bounded by $O(\log n)$.*

Now we are ready for our main theorems in the section.

Theorem 3.6. *Let L be an arbitrary parameterized problem in NP. Then L is in $GC(R(k), P)$ if and only if L is in FPT.*

Proof. (Sketch) Let $L \in FPT$. Then there is a deterministic algorithm M_1 deciding whether $\langle x, k\rangle$ is in L in time $f(k)|x|^c$, where by Lemma 3.3, we can assume that $f(k)$ is an unbounded nondecreasing log-space constructable function and c is a constant independent of k. Moreover, since L is in NP, there is a nondeterministic algorithm M_2 that decides whether $z = \langle x, k\rangle$ is in L in time $q(|z|)$, where q is a polynomial.

Now construct a deterministic algorithm M as follows. Given an input of the form $z\#y$, where $z = \langle x, k\rangle$, the algorithm M first compares $f(k)$ and $|x|$. If $f(k) \leq |x|$ then M simulates the algorithm M_1 on input $z = \langle x, k\rangle$. If $f(k) > |x|$ then M simulates the algorithm M_2 on input $z = \langle x, k\rangle$ following the computation path specified by the string y.

We analyze the complexity of the algorithm M. By Lemma 3.4, $f(k) \leq |x|$ is equivalent to $f^{-1}(|x|) \geq k$. Moreover, according to Lemma 3.5, the function value $f^{-1}(|x|)$ can be computed in time $O(|x|^d)$ for some constant d. Therefore, in time $O(|x|^d) \leq O(|z|^d)$, the algorithm M can check whether $f(k) \leq |x|$. In case $f(k) \leq |x|$, the algorithm M simulates the algorithm M_1 on input $z = \langle x, k\rangle$, which runs in time $f(k)|x|^c \leq |x|^{c+1} \leq |z|^{c+1}$, while in case $f(k) > |x|$, the algorithm M simulates the algorithm M_2 on input z following the computation path specified by y, which is a deterministic computation of running time $q(|z|)$. In summary, the running time of the algorithm M is bounded by a polynomial of $|z|$.

The construction of M shows that the language L is in the class $GC(q(f(k)+\log k), P)$. In fact, for a given input $z = \langle x, k\rangle$, if $f(k) \leq |x|$, the algorithm M can decide whether $z \in L$ without consulting the guessed string y; while in case $f(k) > |x|$, the length $|z|$ of $z = \langle x, k\rangle$ is bounded by $f(k)+\log k$ and the length of the computation path of M_2 is bounded by $q(|z|) \leq q(f(k)+\log k)$. Therefore, a guessed string y of length at most $q(f(k)+\log k)$ is sufficient to specify an accepting computation path of M_2.

The other direction of the theorem is relatively easier. Let L be a problem in $GC(f(k), P)$, where f is a recursive function. Given an input $\langle x, k\rangle$, a deterministic algorithm M simply enumerates all strings y of length $f(k)$ and simulates the verifier on the input $\langle x, k\rangle \# y$. It is easy to see that the running time of the algorithm M is bounded by $g(k)n^c$, where g is a recursive function and c is a constant independent of the parameter k. □ □

Now we consider the structure of the class $W[P]$. The class $W[P]$ can also be characterized by the GC model, as stated in Theorem 3.9. However, the proof technique is very different.

Lemma 3.7. *The problem WCS is in the class* $GC(k \log n, P)$.

Lemma 3.8. $GC(R(k) \log n, P) \subseteq W[P]$.

Proof. (Sketch) Suppose that $L \in GC(f(k) \log n, P)$, where $f(k)$ is a recursive function. By the definition, there is a deterministic algorithm M such that for each instance $z = \langle x, k\rangle$, $z \in L$ if and only if there is a string y, $|y| \leq f(k) \log(|x|)$ such that $z \# y \in L(M)$ and the running time of M is bounded by a polynomial q of $|z|$. We show how the problem L can be uniformly reduced to the problem *WCS*.

Let $z = \langle x, k\rangle$ be an instance of L. Since the running time of M is bounded by $q(|z|)$, in time polynomial in $|z|$, we can construct a circuit C such that C accepts $z \# y$ if and only if the algorithm M does [**?**]. Assigning the input part corresponding to the string z by the value of z gives a circuit C' with $f(k) \log n$ input bits. Now construct another circuit C'' with $f(k)n$ input bits such that the circuit C'' accepts a weight $f(k)$ input vector if and only if the circuit C' has a satisfiable assignment, which is in turn if and only if the algorithm M accepts the input $z \# y$ for some string y of length $f(k) \log n$. It can be shown that the above reduction can be done in time $h(k)n^c$, where h is a recursive function and c is a constant independent of k. This shows that the problem L is reducible to the problem *WCS*. Thus, L is in the class $W[P]$. □ □

Theorem 3.9. *Let L be a parameterized problem in NP. Then L is in the class $GC(R(k) \log n, P)$ if and only if L is in the class $W[P]$.*

Proof. (Sketch) By Lemma 3.8, we only need to show that if L is in $W[P]$ then L is in $GC(R(k) \log n, P)$.

Since the problem L is in NP, there is a nondeterministic algorithm M_0 that decides whether $\langle x, k\rangle$ is in L in time $p(|x|+|k|)$, where p is a polynomial.

Suppose $L \in W[P]$, by definition L is reducible to the problem WCS, that is, there is a deterministic algorithm M_1 running in time $f(k)|x|^c$ that transforms a pair $\langle x, k\rangle$ into a pair $\langle C, g(k)\rangle$ such that $\langle x, k\rangle \in L$ if and only if the circuit C accepts a weight $g(k)$ input vector, here f and g are recursive functions and by Lemma 3.3, we may assume that the function f is unbounded, non-decreasing, and log-space constructable.

By Lemma 3.7, the problem WCS is in the class $GC(k \log n, P)$. Thus, there is a deterministic polynomial time algorithm M_2 such that $\langle C, k\rangle$ is in WCS if and only if there is a string y of length at most $k \log |C|$ and M_2 accepts $\langle C, k\rangle \# y$.

Now we construct a deterministic algorithm M as follows. Given an input of the form $z\#y$, where $z = \langle x, k\rangle$, M first check if $f(k) \leq |x|$. As indicated in the proof of Theorem 3.6, this checking can be done in time $|x|^d \leq |z|^d$ for some constant d. If $f(k) \leq |x|$, then M simulates the algorithm M_1, transforms the pair $z = \langle x, k\rangle$ into an instance $\langle C, g(k)\rangle$ of the problem WCS, and then simulates the algorithm M_2 on input $\langle C, g(k)\rangle \# y$. If $f(k) > |x|$, then M simulates the nondeterministic algorithm M_0 on input $\langle x, k\rangle$ following the computation path specified by y. It is easy to show that $\langle x, k\rangle$ is in L if and only if M accepts an input $\langle x, k\rangle \# y$, where y is a string of length at most $\max\{p(f(k)+|k|), g(k) \log n\}$. Moreover, it can also be shown that the running time of M is bounded by a polynomial of $|z|$. Thus, L is in the class $GC(R(k) \log n, P)$. □ □

Theorem 3.6 and Theorem 3.9 give a strong evidence that the class FPT is a proper subclass of the class $W[P]$ and point out an intrinsic difference among various NP-hard optimization problems. For example, by Theorem 3.6, to find a vertex cover of size k in a graph G, which is a fixed-parameter tractable problem, we only need to guess a string of length $f(k)$, no matter how large the graph G is, where $f(k)$ is a fixed recursive function depending only on the parameter k; while according to Theorem 3.9, to find a weight k satisfying assignment for a circuit C, for which the corresponding problem WCS is complete for $W[P]$, we *must* be able to guess a string whose length is of the form $g(k) \log |C|$ for some recursive function $g(k)$, which increases with the size $|C|$ of the circuit C.

Formally, using these characterizations and padding techniques, we can show that $W[P] = FPT$ implies an unlikely consequence in complexity the-

ory, as stated in the following theorem.

Theorem 3.10. *In the following, (1) and (2) are equivalent, and both implies (3).*
(1) $W[P] = FPT$;
(2) $GC(s(n)\log n, P) \subseteq P$ *for some unbounded non-decreasing function* $s(n)$;
(3) $NTIME(n) \subseteq DTIME(2^{o(n)})$.

4 The structure of the W-hierarchy

In this section, we will characterize the W-hierarchy by the GC model and show a strong evidence that the W-hierarchy does not collapse.

Definition 4.1. *Let* $\mathcal{C}$ *and* $\mathcal{C}'$ *be two classes of parameterized problems. The two classes are equivalent up to the uniform reduction if every problem in class* $\mathcal{C}$ *can be uniformly reduced to some problem in the class* $\mathcal{C}'$ *and vice versa.*

Let Π_t (resp. Σ_t) denote the class of languages accepted by log-time alternating Turing machine of alternation depth at most t, and that must begin with $\wedge$ state (resp. $\vee$ state). For a more careful discussion of this kind of alternating Turing machines, the reader is referred to [**?**, **?**].

Lemma 4.2. *The class* $GC(R(k)\log n, \Pi_{2t})$ *is a subclass of the class* $W[2t]$, *for all* $t \geq 1$.

Proof. (Sketch) The proof is similar to that of Lemma 3.8. Suppose that L is a problem in the class $GC(f(k)\log n, \Pi_{2t})$, where f is a recursive function. Then there is an alternating Turing machine M of alternation depth $2t$, such that $z = \langle x, k\rangle$ is in L if and only if there is a string y of length $f(k)\log(|x|)$ and M accepts the input $z\#y$ in $O(\log|z|)$ space. We show how to reduce the problem L to the problem $WCS(2t, h)$ for some constant h.

Let $z = \langle x, k\rangle$ be an instance of L. By Theorem 3.3 in [**?**], a Π^h_{2t}-circuit C can be constructed in time polynomial in $|z|$ such as the circuit C accepts the input $z\#y$ if and only if the algorithm M does, where y is a string of length $f(k)\log(|z|)$. Assigning the input part of C corresponding to the string z results in a Π^h_{2t}-circuit C' with $f(k)\log(|z|)$ input bits. Now by a technique more delicate than that of Lemma 3.8, we can construct a Π^h_{2t}-circuit C'' of

$f(k)n$ input bits *without increasing the depth* so that the circuit C' has a satisfiable assignment if and only if C'' accepts a weight k input vector. This gives the uniform reduction from the problem L to the problem $WCS(2t, h)$. Thus, L is in the class $W[2t]$. □ □

Lemma 4.3. *For each fixed integer t, the parameterized problem $WCS(2t, h)$ is in the class $GC(k \log n, \Pi_{2t})$, for all integer h.*

Proof. (Sketch) On an input of form $\langle C, k\rangle \# y$, where C is a Π_{2t}^h-circuit and $|y| = k \log |C|$, the log-time alternating Turing machine M of alternation depth $2t$ will interpret the string y as k input gate names and check whether the assignment that assigns all these k inputs 1 and all other inputs 0 is satisfiable. □ □

Theorem 4.4. *The classes $GC(R(k) \log n, \Pi_{2t})$ and $W[2t]$ are equivalent up to the uniform reduction.*

Proof. Follows directly from Lemma 4.2 and Lemma 4.3 and the definitions. □ □

Therefore, in some sense, the class $GC(R(k) \log n, \Pi_{2t})$ and the class $W[2t]$ are of the same fixed-parameter complexity. Surprisingly enough, we show below that all classes $GC(R(k) \log n, \Pi_t)$ are distinct.

Theorem 4.5. *The class $GC(R(k) \log n, \Pi_t)$ is a proper subclass of the class $GC(R(k) \log n, \Pi_{t+1})$ for all $t \geq 1$.*

Proof. Suppose the theorem is not true so that

$$GC(R(k) \log n, \Pi_t) = GC(R(k) \log n, \Pi_{i+1})$$

Let A be a language in the class Π_{t+1}. Fix an integer k_0. Consider the following parameterized problem:

$$L_A = \{\langle x, k_0\rangle \mid x \in A\}$$

Then clearly, $L_A \in GC(f(k) \log n, \Pi_{t+1})$ for some recursive function f (in fact, we can choose $f(k) \equiv 0$). By our assumption, the problem L_A is in the class $GC(g(k) \log n, \Pi_t)$ for some recursive function g. Since the parameter k in the problemL_A is fixed to be k_0, the problem L_A is in the class $GC(g(k_0) \log n, \Pi_t)$. Thus, there is a log-time alternating Turing machine M

of alternation depth t such that for any $z = \langle x, k_0 \rangle$, $z \in L_A$ if and only if there is a string y of length $g(k_0)\log(|x|)$ and M accepts $z\#y$. Construct a new log-time alternating Turing machine M' as follows: given an input x, M first existentially guesses a string y of length $g(k_0)\log n$ then simulates the machine M on input $\langle x, k_0 \rangle \# y$. It is easy to see that M accepts the language A. Moreover, the machine M' has alternation depth $t+1$ and running time is bounded by $O(\log n)$. Thus, the language A is in the class Σ_{t+1}.

Since A is an arbitrary language in Π_{i+1}, we conclude that $\Pi_{i+1} \subseteq \Sigma_{i+1}$. However, this contradicts a result by Sipser [**?**] (see also [**?**] and [**?**]). □ □

Corollary 4.6. *For all $t \geq 1$, the class $GC(R(k)\log n, \Pi_t)$ is a proper subset of the class $GC(R(k)\log n, P)$.*

Another natural question for the W-hierarchy is whether the class $W[P]$ contains all parameterized problems in *NP*. Interesting enough, we show below that this question is closely related to the question whether the classes FPT and $W[P]$ are identical.

Lemma 4.7. *For recursive functions f and g and a polynomial time constructible function $h(n) = \omega(\log n)$, if $GC(f(k)h(n), P) \cap NP = GC(g(k)\log n, P) \cap NP$, then $GC(h(n), P) = GC(\log n, P)$.*

Theorem 4.8. *If $W[P]$ contains all parameterized problems in NP, then $W[P] = FPT$.*

Proof. Let $h(n)$ be any function $= \omega(\log n)$. Consider parameterized problems in $GC(R(k)h(n), P) \cap NP$. By the assumption of the theorem, $GC(R(k)h(n), P) \cap NP \subseteq W[P]$. By Theorem 3.9, $GC(R(k)h(n), P) \cap NP \subseteq GC(R(k)\log n, P) \cap NP$, which implies immediately $GC(h(n), P) \subseteq GC(\log n, P) = P$ by Lemma 4.7, which in turn leads to $W[P] = FPT$ by Theorem 3.10. □ □

5 Subclasses in *FPT*

All known *NP* optimization problems with constant approximation ratio have been shown fixed parameter tractable [**?**] (the proofs for some problems may involve nontrivial techniques used in [**?**]). This strong evidence suggests us to conjecture that approximability implies fixed parameter tractability for optimization problems. On the other hand, there are non-approximable *NP* optimization problems that are fixed parameter tractable. For example,

Longest Path has been shown to have no constant approximation ratio [**?**] while it is fixed parameter tractable [**?**]. Thus, a further refinement of the class FPT is needed to capture the approximability of optimization problems in FPT.

A possible refinement has been considered recently, and a variety of practical problems have been shown to belong to various subclasses of FPT [**?**]. In the following, we discuss a refinement of the class FPT in terms of the GC models.

We first consider the GC models with guessing ability strictly limited.

Theorem 5.1. *(1) Vertex-cover is in $GC(k, P)$;*
(2) MAX-3SAT is in $GC(dk, P)$, where d is a fixed integer;
(3) Varying k-CNF is in $GC(k \log k, P)$.

We point out that all above three problems are approximable and they are in the class $GC(p(k), P)$, for some polynomial p. A contrasting example is the problem *Longest Path*, which can be solved in deterministic $O(2^k k! n)$ time [**?**] and is unknown in the class $GC(p(k), P)$ for any polynomial p. On the other hand, it is known that the problem *Longest Path* is not approximable to a constant ratio in polynomial time [**?**].

We can also define subclasses of FPT by limiting the verifier in the GC model. By carefully examining the proofs of Theorem 1.1 and 2.7 in [**?**], we have

Theorem 5.2. *(1) Vertex-cover is in $GC(k \log k, L)$*
(2) k-LEAF SPANNING FOREST is in $GC(6k(k+1) \log k, L)$
(3) k-LEAF SPANNING TREE is in $GC(6k(k+1) \log k, NL)$

It is interesting to notice that separation of subclasses of $GC(R(k), P)$ defined by limiting the verifiers in the GC model implies separation of the corresponding subclasses in P.

Theorem 5.3. *(1) $GC(R(k), P) = GC(R(k), NL)$ if and only if $P = NL$*
(2) $GC(R(k), NL) = GC(R(k), L)$ if and only if $NL = L$
(3) $GC(R(k), P) = GC(R(k), L)$ if and only if $P = L$

6 Conclusion

We have characterized the classes in the W-hierarchy by the GC models. Our results give strong evidence for that the W-hierarchy does not collapse.

New techniques have been developed to separate the ability of guessing and the power of verifying in computations of practical problems. The class of fixed-parameter tractable problems have been further refined for possible classifications in order to capture the approximability of problems in the class FPT. These results should have important impact to the study of computational optimization problems, in particular for those studies based on the theory of completeness of parameterized problems.