The Measure Hypothesis and Efficiency of Polynomial Time Approximation Schemes

M. Hauptmann[*]

February 11, 2008

Abstract

A polyomial time approximation scheme for an optimization problem X is an algorithm $\mathcal{A}$ such that for each instance x of X and each $\epsilon > 0$, $\mathcal{A}$ computes a $(1+\epsilon)$-approximate solution to instance x of X in time is $O(|x|^{f(1/\epsilon)})$ for some function f. If the running time of $\mathcal{A}$ is instead bounded by $g(1/\epsilon) \cdot |x|^{O(1)}$ for some function g, $\mathcal{A}$ is called an efficient polynomial time approximation scheme. $PTAS$ denotes the class of all NP optimization problems for which a polytime approximation scheme exists, and $EPTAS$ is the class of all such problems for which an efficient polytime approximation scheme exists. It is an open question whether $P \neq NP$ implies the strictness of the inclusion $EPTAS \subseteq PTAS$. Bazgan [2] and independently Cesati and Trevisan [5] gave a separation under the stronger assumption $FPT \neq W[P]$. In this paper we prove $EPTAS \subsetneq PTAS$ under some different assumption, namely existence of NP search problems Π_R with a superpolynomial lower bound for the deterministic time complexity. This assumption is weaker than the NP Machine Hypothesis [15] and hence is implied by the Measure Hypothesis $\mu_p(NP) \neq 0$. Furthermore, using a sophisticated combinatorial counting argument we construct a recursive oracle under which our assumption holds but that of Cesati and Trevisan does not hold, implying that using relativizing proof techniques one cannot show that our assumption implies $FPT \neq W[P]$.

1 Introduction

A polynomial time approximation scheme for an optimization problem X is an algorithm A such that for every instance x of X and every $\epsilon > 0$, A

[*]Dept. of Computer Science, University of Bonn. Email:hauptman@cs.uni-bonn.de

returns a $(1+\epsilon)$-approximative solution $y = A(x, \epsilon)$ in time $O(|x|^{f(1/\epsilon)})$ for some function f. If instead the running time can be bounded by $O(g(1/\epsilon) \cdot |x|^c)$ for some constant c and some function g, algorithm A is called an *efficient polynomial time approximation scheme* (eptas).

NPO denotes the class of all NP optimization problems, $PTAS$ the class of all problems $X \in NPO$ that admit a polynomial time approximation scheme (ptas) and $EPTAS$ all problems in NPO that admit an efficient ptas. Obviously, $EPTAS \subseteq PTAS$. It is an open problem whether the strictness of this inclusion follows from $P \neq NP$.

Cesati and Trevisan [5] and independently Bazgan [2] separate these two classes under the stronger assumption $W[P] \neq FPT$ from Fixed Parameter Complexity. Indeed, they showed the following chain of implications:

$$W[P] \neq FPT \implies EPTAS \neq PTAS \implies FPT \neq XP.$$

Here $FPT \subseteq W[P] \subseteq XP$ are classes of the W-hierarchy introduced by Downey and Fellows [7]. For further details we refer to [8] and to [10]. The assumption $FPT \neq W[P]$ was characterized in terms of bounded nondeterminism properties of NP search problems by Cai and Chen [4]. Cai and Chen [4] proved that $FPT \neq W[P]$ iff for each unbounded polynomial-time computable function $s\colon \mathbb{N} \to \mathbb{N}$, $NP[s(n)\log(n)] \not\subseteq P$, where $NP[f(n)]$ denotes the class of NP problems that can be solved in polynomial time with $O(f(n))$ nondeterministic steps.

We obtain the following results. 1. We separate $PTAS$ from $EPTAS$ under some different assumption, namely existence of an NP search problem Π_R (where R denotes some polynomially balanced, polynomial-time decidable binary relation) and a superpolynomial function $t(n)$ such that Π_R is not solvable in time $O(t(n))$. Under this assumption which we simply call *Assumption (A)*, we construct for each strictly monotone recursive function f an NP optimization U_f such that U_f admits a polynomial time approximation scheme with running time $O(|x|^{f(1/\epsilon)})$ and such that $U_f \notin EPTAS$.

2. Under the assumption $P \neq NP$ we show that for each recursive function $f(n)$, there exists an NPO problem U_f that provides an efficient ptas, but there exists no ptas for U_f with running time $O(|x|^{f(1/\epsilon})$. Furthermore, again assuming $N \neq NP$ we construct another NPO problem U such that $U \in EPTAS$ but no ptas for U has a running time $O(|x|^{f(1/\epsilon)})$ with f being recursive.

Let us point out some connections to the Quantitative Complexity Theory. *Assumption (A)* is a direct implication of the *NP Machine Hypothesis*, which states the existence of an NP machine accepting 0^* such that no 2^{n^ϵ}-time bounded machine can find infinitely many accepting computations.

Jack Lutz defined polynomial measure μ_p [17] and p-dimension $\dim_p$ [18] as generalizations of the classical Lebesgue measure and the Hausdorff dimension. Scaled dimension was introduced by Hitchcock, Lutz and Mayordomo [14]. It was shown by Hitchcock and Pavan [15] that the Measure Hypothesis implies the NP Machine Hypothesis. On the other hand , results from Hitchcock ([13], Theorem 5.2) directly imply that *assumption (A)* already follows from the assumption that for some $i \in \mathbb{Z}$, $\dim_p^{(i)}(NP) = 1$. Here $\dim_p^{(i)}$ denotes the i-th order scaled dimension as introduced by Hitchcock, Lutz and Mayordomo [14].

Furthermore, we construct an oracle X relative to which *assumption (A)* holds but the assumption of Cai and Chen [4] fails. Hence there is no relativizing proof that our assumption implies $FPT \neq W[P]$. The oracle construction involves a sophisticated combinatorial counting argument which is based on ideas from Beigel and Goldsmith [3].

Finally we get a similar separation result for *randomized polynomial-time approximation schemes*, based on an assumption of a superpolynomial lower bound on the randomized time complexity of an NP-search problem.

This paper is based on [11], chapter 4. Some of the results were also presented in [12].

The paper is organized as follows. Preliminaries are given in the next section. Section 3 contains our separation results under assumption $P \neq NP$. In section 4 we separate EPTAS from PTAS under *assumption (A)* stated above. In Section 5 we construct an oracle relative to which assumption (A) holds but $FPT \neq W[P]$ fails. Finally in section 6 we are concerned with the separation $REPTAS \subsetneq RPTAS$ based on a lower-bound assumption which is the analog of assumption (A) in the randomized setting.

2 Preliminaries

Let $\Sigma = \{0,1\}$. A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ is called $f(n)$-balanced if for each $(x,y) \in R$, $|y| \leq f(|x|)$. For such a relation R, let $L_R = \{x \in \Sigma^* | \, \exists \, y \in \Sigma^* \; (x,y) \in R\}$ be the projection of R to the first component. NP can be characterized as the set of all $L \subseteq \Sigma^*$ such that there exists a polynomially balanced polynomial-time decidable relation $R \subseteq \Sigma^* \times \Sigma^*$ with $L_R = L$.

Given such a relation R, Π_R denotes the associated *NP search problem*: Given some $x \in \Sigma^*$, either compute some string π such that $(x,\pi) \in R$ or return "NO" in case no such π exists. $NP[f(n)]$ denotes the class of all NP problems that are solvable in polynomial time with $O(f(n))$ nondeterministic steps - equivalently: the class of all problems L_R such that R is a polynomial-time decidable, $O(f(n))$-balanced binary relation. Especially, $\beta_k := NP[\log^k(n)] := GC(\log^k(n))$ $(k \in \mathbb{N})$ are the levels of the Kintala-Fischer hierarchy [16] which is also called the β-hierarchy. The possibility of downward separation within this hierarchy was invented by Beigel and Goldsmith [3], who proved that using relativizing proof techniques one cannot obtain any strictness result for this hierarchy.

NPO denotes the class of *NP optimization problems*. For the precise definition we refer to [1]. If $X \in NPO$, we call X an *NPO problem*.

3 Separations under Assumption $P \neq NP$

In this section we study the dependence of running times of polynomial-time approximation schemes on the precision parameter ϵ. Recall that for an optimization problem X, to be in NPO means solutions being polynomially bounded in the length of the instance, polynomial time computability of the cost function and polytime checkability whether some string x is a legal instance of the problem and whether string y is a feasible solution to instance x of problem X. Hence in exponentiall time $2^{n^{O(1)}}$ one can solve the problem X to optimality. One may now ask the following question: Are there problems in $PTAS$ for which there does not exist an approximation scheme with recursive dependence of the running time on $\epsilon = 1/n$?

In this section we will give some first results concerning uniformity and efficiency of polynomial time approximation schemes, answering the question affirmatively.

Theorem 3.1. *Unless P=NP, for every recursive function $f\colon \mathbb{N} \to \mathbb{N}$ there is a problem $U \in PTAS$ such that:*

1. *There exists no polynomial time approximation scheme for U with running time $|x|^{O(f(n))}$.*
2. *There exists a polynomial time approximation scheme for U with running time $g(n) \cdot |x|$ for some recursive function g.*

Proof: Let $T_i, i = 1, 2, \ldots$ be an effecive enumeration of Turing machines. Let $U_n, n \in \mathbb{N}$ be the family of APX-complete problems from last section, $p(n)$ the polynomial and T the Turing machine such that T has time complexity bounded by $p(n)$ and for each $n \in \mathbb{N}$ U_n is $p(x)$-bounded and T is a $(1 + 1/n)$-approximation algorithm for U_n. We will construct an NPO problem U with the properties 1 and 2 from the theorem in stages in terms of a lexicographically increasing infinite sequence of strings $x_0, x_1, \ldots, x_n, \ldots$ such that the following holds:

(a) We call $I_n := [x_{n-1}, x_n)$ the *n-th interval.* The problem: Given $x \in \Sigma^*$, compute the number n with $x \in I_n$ is polynomial time solvable.

(b) Optimization Problem U restricted to strings from the interval I_n is equal to U_n.

(c) For each n there exist pairs $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ such that for all $i \in \{1, \ldots, n\}$ $y_{n,i} \in I_n$ and $T_i(y_{n,i}, m_{n,i})$ is not a $(1 + 1/m_{n,i})$-approximation to U_n in time $|y_{n,i}|^{f(m_{n,i})}$.

Stage 0*:* Let $x_0 := 0$.
Stage n (for $n > 0$): Consider the following recursive algorithm for the construction of x_n:

```
Algorithm Construct
Input: $n \in \mathbb{N}$      Output: $x_n \in \Sigma^*$
(1)  If $n = 0$ return $x_0 = 0$ else
          Let $x_{n-1} :=$ Construct$(n-1)$. Find pairs
          $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ by brute force.
(2)  Let $t_n$ be the time used for step (1).
     $x_n := 0^{t_n+1}$. Return $x_n$.
```

Here "brute force" in step (1) means the following: we check pairs (y, m) in lexicographic order starting with $(x_{n-1}, 2)$. For each pair (y, m) the computations $T_i(y, m).i = 1, \ldots, n$ are simulated for at most $|y|^{f(m)}$ steps. If a computation stops within that time, the result is compared to the optimum solution $\mathrm{OPT}_{U_n}(y)$ of instance y for problem U_n, which is computed by brute force, i.e. by trying all strings up to length $p(|y|)$. Note that since U_n is APX-complete, it does not permit a ptas and therefore pairs $(y_{n,i}, m_{n,i}), 1 \leq i \leq n$ exist. In order to show that U is an NPO problem it suffices to prove (a). Given x, we can easily compute n with $x \in I_n$ by simulating for at most $|x|$ steps the construction of $x_0, x_1, \ldots$. Let x_m the last string that was constructed by this process, then $n = m + 1$.
By the diagonal construction U does not provide a ptas with running time $|x|^{f(n)}$. We will now argue that the following algorithm is an efficient ptas for U with time complexity $g(n) \cdot |x|$ for some recursive function g.

```
Algorithm Approximate
Input: $x \in \Sigma^*, n \in \mathbb{N}$      Output: $(1 + 1/n)$-approximate solution $y$
(1)  Compute $m \in \mathbb{N}$ with $x \in I_m$.
(2)  If $m \geq n$ return $T(x)$ otherwise
         Compute an optimum solution $y_x$ to instance $x$ of $U_n$
         by brute force, return $y_x$
```

Obviously algorithm *Approximate*(x, n) computes a $(1 + 1/n)$-approximate solution y_x to instance x of U. For given $n \in \mathbb{N}$, for $x \geq x_{n-1}$ the running time is $p(|x|)$, for $x < x_{n-1}$ it is at most $|x_{n-1} \cdot 2^{|x_{n-1}|} =: g(n)$. Hence we obtain a time bound of $g(n) \cdot |x|$, and obviously g is a recursive function, which completes the proof. □

Theorem 3.2. *If P$\neq$ NP then there exists a problem U $\in$NPO such that*

1. *There is a polynomial time approximation scheme for U with time complexity $g(n) \cdot p(|x|)$ for some polynomial p and some function $g\colon \mathbb{N} \to \mathbb{N}$. (Hence U belongs to the class EPTAS.)*
2. *For every recursive function $f\colon \mathbb{N} \to \mathbb{N}$, U does not have a ptas with time complexity $|x|^{f(n)}$ (hence U does not belong to Uniform-PTAS).*

Proof: We will first describe the main ideas and afterwards give the precise proof: Again we will construct U in stages in terms of an infinite sequence $x_0, x_1, \ldots, x_n, ..$ of strings. As in the proof of the previous theorem, assume $U_n, n \in \mathbb{N}$ to be a family of APX-complete problems with uniform bound $p(x)$ and T a Turing machine with running time bounded by the same polynomial $p(x)$ such that for every $n \in \mathbb{N}$ T is a $(1 + 1/n)$-approximation algorithm for U_n. Assume further that for each n the existence of a polynomial time $(1 + 1/h_n)$-approximation algorithm would imply P=NP, such that the function $n \mapsto h_n$ is recursive. Let $(A_i, T_i, \alpha_i)_{i\in\mathbb{N}}$ be an effective enumerations of the set of triples (A, T, α) where A and T are Turing machines and $\alpha \in \mathbb{N}$.
The idea of our construction is as follows: In stage n assume we have already constructed the sequence up to x_{m-1} for some $m \leq n$. We start simulating the computation $A_i(h_n)$ for $i = 1, \ldots, n$. In every stage we maintain a list $\mathcal{L} = \{(A_i, n_j, C_i)\}$ of actually simulated computations $A_i(hn_j)$, where C_i denotes the current configuration up to which the computation is already simulated. In stage n, after adding the pairs $(A_i, n, C_i^0), 1 \leq i \leq n$ with initial configurations C_i^0 of computations $A_i(h_n)$ to list $\mathcal{L}$, for all entries of the list we simulate one step. If none of the computations stop, we go to step $n+1$. If, say, (A_i, n_j, C_i) stops (or has already stopped before stage n) and (i, j) is the lexicographically first pair with that property then U restricted to interval $I_m = [x_{m-1}, x_m)$ will be defined as U_j in order to satisfy the following constraint

$C_{n,i}$: If $A_i(h_n)$ stops then $T_i(\ , h_n)$ is not a $\left(1 + \frac{1}{h_n}\right)$-approximation algorithm for U in time $\alpha_i \cdot |x|^{A_i(h_n)}$.

Constraint $C_{n,i}$ can be satisfied by instances $x \geq x_{m-1}$ using the properties of U_n. The crucial fact that will enable us to give a ptas for U is that for every $n \in \mathbb{N}$, $U|I_m$ will be set to U_n only finitely many times (namely when one of the computations $A_i(h_n), i = 1, \dots n$ stops. Therefore for every $n \in \mathbb{N}$ after at most finitely many steps U will be defined as U_m for $m \geq n$ and hence $(1 + 1/n)$-approximation is possible.
Let us now give the details. First we describe the construction of problem U in stages.

Initialization: $x_0 := 0, m := 1$, $\mathcal{L} := \emptyset$ (the empty list).
Stage 0**:** Do nothing.
Stage $n > 0$**:** Add $(A_i, n, C_i^0), i = 1, \dots, n$ to list $\mathcal{L}$, where C_i^0 is the initial configuration of the computation $A_i(h_n)$. For every triple (A, n', C) such that C is not a stopping configuration simulate one further step. If there is no triple in $\mathcal{L}$ whose computation has already terminated, stage n is done. Otherwise let (A_i, n_j, C_i) be the lexicographically first such triple, ordered first by the value n_j and then by A_i. Remove (A_i, h_j, C_i) from $\mathcal{L}$. Let $x \geq x_m$ be the first instance such that either $T_i(x, h_{n_j})$ does not terminate within time $\alpha_i \cdot |x|^{A_i(h_{n_j})}$ or $T_i(x, h_{n_j})$ is not an $(1 + 1/h_{n_j})$-approximate solution to instance x of U_{n_j}. Such x exists since $(1+1/h_{n_j})$-approximation to U_{n_j} in polynomial time is assumed to be NP-hard. Let t_n be the time needed to recursively compute x_{m-1} by performing stages 0 to $n-1$ and to find x by brute force. Let $x_m := 0^{t_n+1}$.
End of Construction

As before, there is a linear time algorithm which, for given x, computes numbers m, n with $x_{m-1} \leq x < x_m$ and $U|[x_{m-1}, x_m) = U_n$. Hence U is an NPO problem. Consider the following algorithm:

```
Algorithm Approximate
Input: instance $x \in \Sigma^*$ of $U$, $n \in \mathbb{N}$
Output: $(1 + 1/n)$-approximate solution $y$ for $x$
   (1)  Compute $m \in \mathbb{N}$ with $x_{m-1} \leq x < x_m$.
        Compute $n' \in \mathbb{N}$ with $U|[x_{m-1}, x_m) = U_{n'}$.
   (2)  If $n' \leq n$ return $T(x)$ else
            Compute an optimum solution $y^*$ to instance $x$ of $U_{n'}$.
            Return $y^*$.
End.
```

Obviously **Approximate**(x, n) is a $(1 + 1/n)$-approximate solution for instance x of U. Since for every $n' < n$ only tuples (A_i, n', C) with $i \leq n'$

are added to list $\mathcal{L}$ and only those tuples can cause $U|[x_{m-1}, x_m) = U_{n'}$, it follows that for all but finitely many m, $U|[x_{m-1}, x_m) = U_{n''}$ for $n'' \geq n$. Hence the running time of **Approximate** can be bounded by $g(n) \cdot p(|x|)$ for some function g.

Now assume that $f\colon \mathbb{N} \to \mathbb{N}$ is some recursive function and there exists a ptas for U with running time bounded by $c \cdot |x|^{f(n)}$for some constant c. Let $i \in \mathbb{N}$ be such that this ptas be given by Turing machine T_i, f is computed by Turing machine A_i and $c = \alpha_i$. Then in stages $n \geq i$ computations $A_i(h_n)$ are started. Since A_i is total and by the priority rule in the construction of U, for every $n \in \mathbb{N}$ there exists an interval $I = [x_{m-1}, x_m)$ such that $U|I = U_n$ and I contains an x such that $T_i(x, h_n)$ is not a $(1 + 1/h_n)$-approximate solution to U_n in time bounded by $\alpha_i \cdot |x|^{A_i(h_n)}$, a contradiction.

$\square$

4 Separating *PTAS* from *EPTAS*

In this section we will prove $EPTAS \subsetneq PTAS$ under the following assumption.

> **Assumption (A):** There is a language $L \in NP \backslash P$ and a polynomially bounded binary relation $R \subseteq \Sigma^* \times \Sigma^*$ with $L = L_R$ such that $\Pi_R \in DTIME(T(n)) \setminus DTIME(t(n))$ for two polynomial time computable superpolynomial functions $t(n), T(n)$ such that $t(n)^2 = O(T(n)$.

Here a function $f(n)$ is called *superpolynomial* if for each $c > 0$, $f(n) = \omega(n^c)$. Concerning polynomial-time approximation schemes, their running times are bounded by $O(|x|^{f(\epsilon)})$, where x denotes the instance and ϵ the accuracy parameter.

Our main result is stated in the following theorem.

Theorem 4.1. *Under assumption (A), for every strictly monotone recursive function $f\colon \mathbb{N} \to \mathbb{N}$ with $f(1) = 1$ there is an NPO problem U_f such that*

1. *U_f has a ptas with running time $O(|x|^{f(1/\epsilon)+1})$.*
2. *For every monotone increasing function $g\colon \mathbb{N} \to \mathbb{N}$ and every $\alpha > 0$, U_f does not have a ptas with running time $g(1/\epsilon) \cdot |x|^{\alpha}$.*

In particular, this implies $EPTAS \neq PTAS$.

In order to prove Theorem 4.1 we will first define a family of NPO problems $U_n, n \in \mathbb{N}$ and then construct U_f in stages in terms of problems

U_n, such that in stage n we diagonalize against the first n Turing machines being efficient approximation schemes for problem U_f. The crucial point will be that although problems U_n will be approximable with better and better ratio (with n increasing), for every fixed $\epsilon > 0$ the time complexity of approximation within $1+\epsilon$ will stay the same.

We will use the following convenntions. First, we only consider accuracy parameters $\epsilon = 1+\frac{1}{n}, n \in \mathbb{N}$. In this sense, problem U_f will have a ptas with running time $O(|x|^{f(n)})$ for $\epsilon = 1/n$, but there exists no ptas with running time $g(n)\cdot|x|^\alpha$ for U_f. Second, we will consider binary relations $R \subseteq \Sigma^* \times \Sigma^*$ equivalently as functions $R\colon \Sigma^* \times \Sigma^* \to \{0,1\}$ (where $R(x,y) = 1$ means $(x,y) \in R$).

Let $f\colon \mathbb{N} \to \mathbb{N}$ be some monotone increasing recursive function as above such that $f(1) = 1$. Let $A \subseteq \Sigma^*$ be some problem in NP, T_A some deterministic TM and p_A some polynomial such that

1. For all $x \in \Sigma^*$, $x \in A$ iff there is some string π of length $|\pi| \leq p_A(|x|)$ such that $T_A(x,\pi)$ accepts in time $p_A(|x|)$.
2. For $R = \{(x,\pi) \mid T_A(x,\pi) \text{ accepts}\}$, the associated NP search problem Π_R is solvable in time $T(n)$ but not in time $t(n)$.

We define a family $U_n, n \in \mathbb{N}$ of optimization problems as follows:

Definition of U_n:
Instances of U_n: $X = (x_1, \ldots, x_n, 0^k)$ such that $k \geq 1$, $x_1, \ldots, x_n \in \Sigma^*$ and

$$t(|x_j|) \;\leq\; |X|^{f(2^j)}, \qquad j = 1, \ldots, n-1. \tag{1}$$

Solutions: $\pi = (\pi_1, \ldots, \pi_n)$ with $\pi_j \in \Sigma^*$, $|\pi_j| \leq p_A(|x_j|), j = 1, \ldots, n$

Costs: $$c_n(X,\pi) \;=\; 2^n + \sum_{i=1}^{n} R(x_i,\pi_i)\cdot 2^{n-i} \in \left[2^n, 2^n + 2^{n+1} - 1\right].$$
End of definition.

Here 0^k serves as a padding string, and $T_A(x_i,\pi_i) \in \{0,1\}$ denotes the result of computation of machine T_A on input x_i, π_i (1 for accept, 0 for reject). The main properties of problems U_n are stated in the next two lemmas.

Lemma 4.1. *For all $n \in \mathbb{N}$ U_n is an NPO problem. There exists a two-variate polynomial $q(x, y)$ such that for all $n \in \mathbb{N}$ U_n is $q(x, t_f(n))$- time bounded, where $t_f(n)$ is a function such that f is computable in time $t_f(n)$.*

Proof: Function $f(n)$ can be computed in time $t_f(n)$. The conditions (1) are equivalent to $\log(t(|x_j|)) \leq f(2^j) \cdot \log(|X|), j = 1, \dots, n-1$ and therefore can be checked in time $t_f(n) \cdot n \cdot \text{poly}(|X|)$. In time $O(n \cdot p_A(|X|))$ one can check whether a given string $\pi = (\pi_1, \dots, \pi_n)$ is a solution to X. The cost $c_n(X, \pi)$ can be computed in time $O(n \cdot p_A(|X|))$. □

Lemma 4.2. *Under Assumption (A), problem U_n has the following properties.*

1. *For $j = 1, \dots n-1$ there is a $\left(1 + 2^{-j}\right)$-approximation algorithm for U_n with running time $|x_1| + |x_2|^{f(2^2)} + \dots + |x_j|^{f(2^j)} \leq n \cdot |X|^{f(2^j)} = O\left(|X|^{f(2^j)}\right)$ (n being fixed).*
2. *For $j = 1, \dots, n-1$, every approximation algorithm $\mathcal{B}$ for U_n with running time $o\left(|X|^{f(2^j)}\right)$ has approximation ratio at least $1 + 2^{-(j+2)}$.*
3. *There is no polynomial time approximation algorithm for U_n with ratio better than $1 + 2^{-(n+2)}$.*

Proof. This basically follows from the construction of U_n, using the fact that the NP search problem Π_R cannot be solved deterministically in time $t(n)$. Assume $\mathcal{A}$ is some algorithm which for each instance $X = (x_1, \dots, x_n, 0^k)$ of U_n computes a feasible solution $\pi_X = (\pi_1, \dots, \pi_n)$. For $i \in \{1, \dots, n\}$ we say $\mathcal{A}$ *answers x_i correct* iff for every instance $X = (x_1, \dots, x_n, 0^k)$ of U_n, if $x_i \in A$ then $T_A(x_i, \pi_i)$ accepts (i.e. $\mathcal{A}$ constructs a proof π_i for the fact $x_i \in A$). We will show:

(a) In time $t(|x_1|) + t(|x_2|) + \dots + t(|x_j|) \leq n \cdot |X|^{f(2^j)}$ one can answer $x_1, \dots, x_j$ correct, and every algorithm answering $x_1, \dots x_j$ correct has approximation ratio at most $1 + 2^{-j}$.

(b) For $j \in \{1, \dots, n\}$ every approximation algorithm that does not answer x_j correct has approximation ratio at least $1 + \frac{1}{2^{j+2}}$.

(c) For $j = 1, \dots, n-1$: In running time $o\left(|X|^{f(2^j)}\right)$ an algorithm cannot answer x_j correct.

(d) An algorithm with polynomial running time cannot answer x_n correct.

Obviously, from (a)-(d) the lemma follows.
Proof of (a): The time bound for answering $x_1, \dots, x_j$ correct follows directly from the definition of U_n and the assumption about the time complexity of $W(A, T_A)$. Assume now that $\pi = (\pi_1, \dots, \pi_n)$ is a feasible solution for instance X such that $x_1, \dots, x_j$ are answered correct. Let

$$C(j) \ := \ 2^n + \sum_{i=1}^{j} [T_A(x_i, \pi_i)] \cdot 2^{n-i} \in \left[2^n, 2^n + 2^{(n-j)} \cdot (2^{j+1} - 1) \right]$$

be the contribution of $\pi_1, \dots, \pi_j$ to the solution cost. Then the approximation ratio $R_{U_n}(X, \pi)$ of solution π for instance X of problem U_n is

$$\begin{aligned} R_{U_n}(X, \pi) &= \frac{\mathrm{opt}_{U_n}(X)}{c_n(X, \pi)} \leq \frac{C(j) + 2^{n-j-1} + \ldots + 2 + 1}{C(j)} \\ &= 1 + \frac{2^{n-j-1} + \ldots + 2 + 1}{C(j)} \leq 1 + \frac{2^{n-j} - 1}{2^n} \leq 1 + 2^{-j} \end{aligned}$$

Proof of (b): Assume $\mathcal{B}$ is some approximation algorithm for U_n such that infinitely often, $\mathcal{B}$ answers x_j incorrectly. We compute a lower bound on the approximation ratio of $\mathcal{B}$: Let X be an instance such that x_j is answered incorrect, i.e. $x_j \in A$ and $T_A(x_j, \pi_j) = 0$ for $\mathcal{B}(X) = \pi = (\pi_1, \dots, \pi_n)$. Let $\pi^* = (\pi_1^*, \dots, \pi_n^*)$ be an optimum solution to instance X of U_n of cost $c_n(X, \pi^*) = 2^n + 2^{n-j} + R^*$. Let $R := \sum_{i \neq j} T_A(x_i, \pi_i) \cdot 2^{n-i}$. Obviously $c_n(X, \pi) = 2^n + R$ and $R \leq R^* \leq 2^{n+1} - 2^{n-j} - 1$. Then the approximation ratio $R_{U_n}(X, \mathcal{B}(X))$ of the solution $\mathcal{B}(X)$ for instance X of U_n constructed by algorithm $\mathcal{B}$ is

$$\begin{aligned} R_{U_n}(X, \mathcal{B}(X)) &= \frac{2^n + 2^{n-j} + R^*}{2^n + R} \ \geq \ 1 + \frac{2^{n-j}}{2^n + R} \\ &\geq \ 1 + \frac{2^{n-j}}{2^n + 2^{n+1} - 2^{n-j} - 1} \ \geq \ 1 + \frac{1}{2^{j+2}} \end{aligned}$$

Proof of (c): We give a reduction from the NP search problem Π_R to U_n: Using the padding property of U_n, for each string $x \in \Sigma^*$ we can construct in polynomial time an instance X of U_n such that $x = x_j$ and

$$|X| \in \left[\sqrt[f(2^j)]{T(|x|)}, \ \sqrt[f(2^j)]{T(|x|)} + 1 \right].$$

Now assume there is some algorithm $\mathcal{A}$ with running time $o\left(|X|^{f(2^j)}\right)$ such that for each instance $X = (x_1, \dots, x_n, 0^k)$ of U_n, $\mathcal{A}$ computes a feasible solution $(\pi_1, \ldots, \pi_n)$ for instance X of U_n such that x_j is answered correctly. Then using algorithm $\mathcal{A}$ we would solve the NP search problem Π_R in time

$$o\left(|X|^{f(2^j)}\right) = o\left(f\left(2^j\right) \cdot t(|x|)\right) = o(t(|x|)),$$

a contradiction.
Proof of (d): Such an algorithm could be used to solve the NP search problem Π_R in polynomial time (note that for an instance X of U_n, $|x_n| = \Theta(|X|)$ is allowed), in contradiction to the assumption. This completes the proof of the lemma. □

Proof of Theorem 4.1: We will construct an NPO problem U_f in stages, using a monotone increasing sequence of strings $x_0, x_1, \ldots, x_n, \ldots$. U_f restricted to the n-th interval $I_n := [x_{n-1}, x_n)$ will be equal to U_n. Let $(T_i, c_i, \alpha_i), i \in \mathbb{N}$ be some effective enumeration of triples (T_i, c_i, α_i) where T_i is a Turing machine, $c_i > 0$ some constant and $\alpha_i \in \mathbb{N}$ such that every such triple (T, c, α) occurs infinitely often. We will construct $x_1, \ldots, x_n, \ldots$ such that for every $n \in \mathbb{N}$ the following requirement (C_n) is satisfied:

(C_n) For $j = 1, \ldots, n$ both (a) and (b) hold.

(a) For $m = 1, \ldots, n-1$: If $f(2^m) \geq \alpha_j + 1/n$ then there exists some $y_{j,m} \in I_n$ such that $T_j(y_{j,m}, 2^{m+3})$ is not a $(1 + 2^{-(m+3)})$-approximate solution to instance $y_{j,m}$ of U_f in time bounded by $c_j \cdot |y_{j,m}|^{\alpha_j}$.

(b) There is some $z_j \in I_n$ such that $T_j(z_j, 2^{n+3})$ is not a $(1+2^{-(n+3)})$-approximate solution to instance z_j of U_f in time $c_j \cdot |z_j|^{\alpha_j}$.

Construction of U_f:
Stage 0: $x_0 := 0$.
Stage $n > 0$: Compute $f(2^1), \ldots, f(2^n), f(2^{n+1})$. By Brute Force find the lexicographically first strings $y_{j,m}, j = 1, \ldots, n, m \in \{1, \ldots, n-1\}$ with $f(2^m) \geq \alpha_j + 1$ and z_j with $y_{j,m}, z_j \geq x_{n-1}$ such that (C_n) becomes true. Such pairs exist because of properties 2 and 3 from Lemma 4.2. Let t_n be the time needed to compute $f(j), 1 \leq j \leq n$ and to find strings $y_{j,m}$ and z_j. Let $x_n := 0^{t_n+1}$ and $U_f|I_n := U_n$.
End of construction.

U_f is an NPO problem: There is a linear time algorithm to compute for given $x \in \Sigma^*$ the number $n \in \mathbb{N}$ such that $x \in I_n$, by computing the number

$n-1$ in at most $|x|$ steps. Since in stage $n-1$ we compute $f(n)$, $|x| \geq t_f(n)$, and therefore using Lemma 4.1 we conclude that U_f is bounded by some polynomial $p(|x|)$ and hence an NPO problem.

U_f admits a polynomial time approximation scheme: For given x and n, in order to compute a $(1 + 1/n)$-approximate solution to instance x of U_f we first compute the number $m \in \mathbb{N}$ with $x \in I_m$. If $2^{m-1} \geq n$ we compute a $(1 + 2^{-j})$-approximate solution for $j = \lceil \log(n) \rceil$ in time $|x|^{f(2^j)} \cdot m \leq |x|^{f(2^j)+1}$. Otherwise we compute an optimum solution y^* to instance x by brute force. Since $2^{m-1} < n$ for only finite many m (and therefore finite many x) and $2^{\lceil log(n) \rceil} < 2n$, the running time is bounded by $O\left(|x|^{f(2n)+1)}\right)$ and hence by $O\left(|x|^{f(2n)+1}\right)$.

$U_f \notin EPTAS$: Assume U_f admits a ptas T with running time $g(n) \cdot |x|^\alpha$ for some function g and some constant α. Then for every $n \in \mathbb{N}$ there exists some $i \in \mathbb{N}$ such that $T_i = T, \alpha_i = \alpha$ and $c_i = g(n)$. But for $m \geq n+3$, in I_m there exist $y \in I_m$ such that either $T_i\,(y, 2^m)$ is not a $(1 + 2^{-m})$-approximate solution for instance y of U_f or $T_i\,(y, 2^m)$ does not stop after at most $c_i \cdot |y|^{\alpha_i}$ steps, a contradiction ! This proves the claim and hence Theorem 4.1. $\square$

5 Oracle Constructions

We will now construct an oracle X relative to which Assumption (A) holds but $FPT \neq W[P]$ doesn't. Recall that in [4] $FPT \neq W[P]$ was shown being equivalent to

> *Assumption (B)* For every unbounded polynomial-time computable function $s\colon \mathbb{N} \to \mathbb{N}$, $NP[s(n)\log(n)] \not\subseteq P$.

We start by defining the appropriate relativized classes.

Definition 5.1. *For $X \subseteq \Sigma^*$ and $f\colon \mathbb{N} \to \mathbb{N}$,*

$$NP^X[f(n)] \;:=\; \Big\{L \subseteq \Sigma^* \,|\, \textit{there exists an } O(f(n))\textit{-balanced binary relation } R \subseteq \Sigma^* \times \Sigma^* \textit{ such that } L = L_R \textit{ and } R \in P^X\Big\}$$

It is easy to construct an oracle making assumption (B) false, namely by taking a sufficiently powerful oracle.

Theorem 5.1. *For every polynomial-time computable unbounded function $s(n)$ there exists a recursive set $X \subseteq \Sigma^*$ such that $NP^X[s(n)\log(n)] \subseteq P^X$.*

Proof. Let X be some $DTIME\left(n^{O(s(n))}\right)$-complete set, then the inclusion obviously holds. For a precise argument (yielding a slightly more general result) see the proof of Lemma 5.1 below. $\square$

In order to construct an *oracle relative to which assumption (B) becomes true*, one needs to diagonalize against polynomially bounded oracle machines accepting inputs (x, y) with $|y|$ bounded by $s(|x|)\log(|x|)$ for some nonconstant polytime computable function $s(n)$. We can enumerate polynomially bounded TMs M_i computing functions $s_i(n)$, but by no means we know in advance which of these functions is bounded. Hence we will guess $s_i(n)$ being bounded by some constant C, and if this guess was incorrect, then we will observe the incorrectness after finite amount of time, namely find some n such that $s_i(n) > C$. We perform such a guessing process, and each time a violation occurs, we increase the constant C and perform one diagonalization step against $NP^X[s_i(n)\log(n)]$ being equal to P^X. We obtain:

Theorem 5.2. (An Oracle relative to which Assumption (B) holds) *There existst some recursive set $X \subseteq \Sigma^*$ such that for every unbounded polynomial-time computable function $s(n)$, $NP^X[s(n)\log(n)] \not\subseteq P^X$.*

We will now separate assumptions (A) and (B) by an oracle construction.

Theorem 5.3. (An Oracle relative to which (A) is true but (B) is false) *There exists a recursive set $X \subseteq \Sigma^*$ such that (1) and (2) hold.*

(1) $NP^X[\log^2(x)] \subseteq P^X$

(2) For the relation $R_X := \{(x, y) | x = 0^n, n \in \mathbb{N},\ |y| = 2n,\ y \in X\}$,

$$\Pi_{R_X} \in DTIME^X\left(2^{n^3}\right) \setminus DTIME^X\left(2^n\right).$$

Proof Idea: Let $\tilde{X}$ be some $DTIME\left(n^{O(\log n)}\right)$-complete problem with respect to polynomial-time Karp reductions. Then $X' := \{1xx | x \in \tilde{X}\}$ is $DTIME\left(n^{O(\log n)}\right)$-complete as well. We let $X = X' \cup X''$ where X'' consists of strings of even length only. We use X'' for a diagonalization process in order to assure (2).

The following two lemmas indicate how powerful the set may be in order to make (1) become true.

Lemma 5.1.
For every $DTIME\left(n^{O(\log n)}\right)$-complete problem Y, $NP^Y[\log^2(n)] \subseteq P^Y$.

Proof. Let R be $c \cdot \log^2(n)$ - balanced for some $c > 0$ with $R \in P^Y$. Let $R = L(M, Y)$ for some polytime bounded oracle machine M. We have to show that $\Pi_R \in FTIME^Y\left(n^{O(1)}\right)$. For input x of length n there are less than $n \cdot 2^{c \cdot \log^2(n)} = n \cdot n^{c \cdot \log(n)} = n^{O(\log(n))}$ strings of length bounded by $c \cdot \log^2(n)$, hence a straight forward algorithm will enumerate all such strings and for each of them ask the question to oracle machine M with oracle Y. The running time of this oracle algorithm is bounded by

$$\underbrace{p(n + c \cdot \log^2(n))}_{\text{running time of } M} \cdot \underbrace{n^{O(\log(n))}}_{\sharp\text{strings of that length}} = n^{O(\log(n))},$$

If we replace the oracle questions by calls of a $n^{O(\log(n))}$-time bounded algorithm M_Y for Y, the total running time can be bounded by

$$n^{O(\log(n))} \cdot \underbrace{q(n)^{O(\log(q(n)))}}_{\text{time for one call of } M_Y} = n^{O(\log(n))}$$

where $q(n) := p(n + c \cdot \log^2(n))$ is polynomially bounded. Hence we can reduce this problem to Y (Y is $DTIME(n^{O(\log(n))})$- complete) and obtain $L \in P^Y$. □

Lemma 5.2. *For every $DTIME\left(n^{O(\log n)}\right)$-hard problem Y which is complete for a class $\mathcal{C}$ of languages such that $DTIME(t(n)) \subseteq \mathcal{C}$ implies $DTIME\left(n^{O(\log(n))} \cdot t\left(n^{O(1)}\right)\right) \subseteq \mathcal{C}$, $NP^Y[\log^2(n)] \subseteq P^Y$.*

Proof. Let $L \in NP^Y[\log^2(n)]$ and $Y \in DTIME(t(n))$. We perform the complete-enumeration oracle algorithm as in the previous proof and obtain a running time bounded by

$$\underbrace{n^{O(\log(n))}}_{\sharp\text{ strings to be enum.}} \cdot \underbrace{p(n + c \cdot \log^2(n))}_{\text{single call of the Oracle TM for } L} \cdot \underbrace{t(p(n + c \cdot \log^2(n)))}_{\text{solving one instance of } Y}$$

which is bounded by $n^{O(\log(n))} \cdot t(n^{O(1)})$. Hence $L \in \mathcal{C}$, and L is polynomial-time reducible to Y, therefore $L \in P^Y$. □

Now if we take an arbitrary set $X'' \subseteq \bigcup_{n \in \mathbb{N}} \Sigma^{2n}$ which is complete for such a class $\mathcal{C}$, the disjoint union $X := X' \cup X''$ still satisfies the conditions in Lemma 5.2, hence condition (1) from Theorem 5.3 will still hold.

Proof of Theorem 5.3. Let $X = X' \cup X''$ where X' consists of odd-length strings only and X'' consists of even-length strings only. We choose

$$C^X := \{(i, x, 0^s) | M_i \text{ on input } x \text{ with oracle } X \text{ accepts within } s \text{ steps and } O(\log^2(n)) \text{ nondeterministic steps}\}.$$

Then C^X is $\leq_m^p$-complete for $NP^X[\log^2(n)]$. Let $p(n)$ be a polynomial such that $C^X \in NTIME[\log^2(n)](p(n))$ (i.e. solvable in time $p(n)$ with $O(\log^2(n))$ nondeterministic steps). We will encode C^X into X' in a polynomial manner and use X'' for diagonalisation in order to assure (2). The encoding will be as follows: For all strings $x \in \Sigma^*$,

$$x \in C^X \quad \Longleftrightarrow \quad 1^{p(|x|)^2}\, 0\, x \ \in X.$$

We are now ready to construct X. Let $M_1, \ldots, M_n, \ldots$ denote an enumeration of $O(2^n)$-time bounded Oracle machines such that without loss of generality M_i is $i \cdot 2^n$ - time bounded.

```
**Construction of $X$ in stages**

/⋆ At the end of stage $s$ $X$ is defined up to                     ⋆/
/⋆ strings of length $n_s$. During stage $s$ diagonalization        ⋆/
/⋆ against $L(M_i) = L_X$ is performed by choosing                  ⋆/
/⋆ $n > n_{s-1}$ and assuring $0^n \in L(M-i)\Delta L_X$             ⋆/

Stage 0: $n_0 := 0, \quad X := \emptyset$

Stage $s > 0$:
  /⋆ $X$ is defined up to length $n_{s-1}$ ⋆/
  Choose $n > n_s$ to be a power of 2 and sufficiently large.
  Define $X$ up to length $l := n^3 - 1$.
  Define $C^X$ up to strings $y$ with $p(|y|)^2 + 1 + |y| \leq n^3 - 1$.

  /⋆ $L_X(1^n)$ depends on strings of length $n^3$. ⋆/

  Compute $M_s(1^n, X)$.

  If $M_s(1^n, X)$ accepts then
      Let $y$ be a legal string of length $n^3$.
      $X := X \cup \{y\}$.

  $n_s := \max\{l, s \cdot 2^n\}$
  Freeze $X$ up to length $n_s$.
  Freeze $C^X$ up to strings of length $m$ with $p(m)^2 + 1 + m \leq n_s$.
End of Stage $s$.
```

It remains to define which strings y of length n^3 are **legal** and what it means to choose n **sufficiently large**. The initial idea is simply to pick a string y

which was not asked by the computation $M_s(x,X)$. Such string of length n^3 exists since M_i has running time bounded by $s \cdot 2^n$ and there are 2^{n^3} strings of length n^3 available (we choose n such that $s \cdot 2^n < 2^{n^3}$). The problem is that the encoding of C^X into X may depend on this choice, and the result of $M_s(x,X)$ may in turn depend on this coding and so on.

We accomplish this difficulty by using an approach from Beigel and Goldberg: We show that due to the way the coding of C^X into X is arranged, there are more strings of length n^3 available than are influencing the coding of C^X into X in the range up to length $s \cdot 2^n$:

$M_s(x,X)$ has running time bounded by $s \cdot 2^n$ and therefore may ask at most $s \cdot 2^n$ oracle questions about strings of length bounded by $s \cdot 2^n$.

Due to the encoding, these strings may encode strings from C^X of length bounded by $\sqrt{s \cdot 2^n} = \sqrt{s} \cdot 2^{n/2}$. Each of these encoding strings w in turn may depend on at most

$$p(|w|) \cdot 2^{\log^2(|w|)} = p\left(s^{2^{-1}} \cdot 2^{n/2^1}\right) \cdot 2^{\log^2\left(s^{2^{-1}} \cdot 2^{n/2^1}\right)}$$

strings of this smaller length and so on.

Since X is already fixed up to length $l-1$, the recursion can be cut off at strings of length $\leq l-1$. Therefore the number of such terms is bounded by

$$\log\log(s \cdot 2^n) - \log\log(l-1) \leq \log(\log s + n) - \log\log(n) \leq c \cdot \log(n) = \Theta(\log(n))$$

for some constant $c > 0$. Hence the total number of strings being influenced by the choice of y is bounded by

$$\begin{aligned} & s \cdot 2^n \cdot \prod_{i \leq c \cdot \log(n)} p\left(s^{2^{-i}} \cdot 2^{n/2^i}\right) \cdot 2^{\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right)} \\ = \; & s \cdot 2^n \cdot \prod_{i \leq c \cdot \log(n)} \left(s^{2^{-i}} \cdot 2^{n/2^i}\right)^{O(1)} \cdot 2^{\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right)}. \end{aligned}$$

Since $\log^2\left(s^{2^{-i}} \cdot 2^{n/2^i}\right) = \left(2^{-i} \cdot \log(s) + \frac{n}{2^i}\right)^2 = O(n^2)$, the total number of strings is bounded by $s \cdot 2^n \cdot O(\log(n)) \cdot 2^{O(n)} \cdot 2^{O(n^2)} = 2^{O(n^2)}$, supposed we choose s sufficiently small compared to n, i.e. choose n large, namely $s = 2^{\log s}$, $\log(s) = O(n^2)$.

Hence we find a string y of length n^3 such that adding y to X does not change the result of computation $M_s(x,X)$, therefore the diagonalization step is well-defined. This completes the proof of Theorem 5.3. $\square$

6 Randomized Approximation Schemes: RPTAS versus REPTAS

In this section we consider the question of efficiency for randomized approximation schemes. Recall that a *randomized polynomial-time approximation scheme* $\mathcal{A}$ for an optimization problem X is a probabilistic approximation algorithm $\mathcal{A}$ for X such that

$$Pr\{\text{A.R. of } \mathcal{A}(x,\epsilon) \text{ is at most } 1+\epsilon\} \;\geq\; \frac{3}{4}$$

and the *randomized running time* of algorithm $\mathcal{A}$ is

$$O\left(|x|^{f(1/\epsilon)}\right)$$

for some function $f\colon \mathbb{N} \to \mathbb{N}$. $\mathcal{A}$ is called an *efficient polynomial-time randomized approximation scheme* if the randomized running time is bounded by

$$O\left(f\left(\frac{1}{\epsilon}\right) \cdot |x|^{O(1)}\right)$$

for some function $f\colon \mathbb{N} \to \mathbb{N}$. Let *RPTAS* denote the class of all NP optimization problems for which a randomized polynomial-time approximation scheme exists. *REPTAS* denotes the class of all *NPO* problems that provide an efficient polynomial-time randomized approximation scheme exists.

In the following lemma, the case of *efficient probabilistic approximation schemes* where the *error probability* is *bounded for each fixed* ϵ is considered.

Lemma 6.1. *Assume $\mathcal{A}$ is a probabilistic approximation scheme for NPO problem X, which means*

$$Pr\{\textit{A.R. of } \mathcal{A}(x,\epsilon) \textit{ is at most } 1+\epsilon\} > \frac{1}{2}$$

Assume furthermore that there are functions $f\colon \mathbb{N} \to \mathbb{N}$ and $e\colon \mathbb{N} \to [0,1/2)$ and $\alpha > 0$ such that e is recursive and

$$Pr\{\textit{A.R. of } \mathcal{A}(x,n) \textit{ is at most } 1+\tfrac{1}{n} \textit{ in time bounded by } f(n)\cdot|x|^{\alpha}\} \geq 1-e(n)$$

for every x and n. Then $X \in$ REPTAS.

Proof: We can decrease the error probability to at most $1/4$ by applying the δ-Lemma for Monte Carlo machines to each n: Compute $e(n)$, then the δ-Lemma gives a computable bound $k = k(n)$ such that repeating $\mathcal{A}(\cdot, n)$ k

times and making majority decision decreases the error probability to 1/4. The running time is bounded by $k(n) \cdot f(n) \cdot |x|^{\alpha}$, which proves the lemma. $\square$

We are now going to separate the classes *REPTAS* and *RPTAS* assuming the existence of polynomially balanced relations in P (*NP search problems*) for which the associated functional problems provides a *superpolynomial lower bound on the randomized time complexity* (assumption (A")) in theorem 6.1 below). This is basically the variant of assumption (A') where deterministic time complexity is replaced by randomized time complexity.

Theorem 6.1. *Assume there exists a polynomially balanced binary relation $R \subseteq \Sigma^* \times \Sigma^*$ such that $R \in P$ and such that for the associated NP search problem Π_R, $\Pi_R \in RFTIME\left(t^2(n)\right) \setminus RFTIME(t(n))$ for some superpolynomial polynomial-time computable function $t(n)$* **(Assumption A").** *Then there exists an optimization problem $U_R \in NPO$ such that*

$$U_R \in RPTAS \setminus REPTAS.$$

Proof: Assume R to be as in assumption (A"), and let R be $p(n)$-balanced for some polynomial $p(n)$. Let $T(n) := t^2(n)$. We define problem $U_{R,n}$ as follows:

Instance: $X = (x_1, \ldots, x_n, 0^k)$ such that

$$T(|x_j|) \leq |X|^{2^j}, \; 1 \leq j \leq n-1.$$

Solution: $\pi = (\pi_1, \ldots, \pi_n)$ such that

$$|\pi_j| \leq p(|x_j|), \; 1 \leq j \leq n-1$$

Cost: $\mathrm{cost}_n(X, \pi) = 2^n + \sum_{j=1}^{n} R(x_j, \pi_j) \cdot 2^{n-j}$

Problems $U_{R,n}$ have the following properties:

(1) There exists a two-variate polynomial $q(n,m)$ such that $U_{R,n}$ is a *$q(n,m)$-bounded NP optimization problem* (recall that this means for given x, y in time $q(n, |x|)$ we can check whether x is a valid instance of problem $U_{R,n}$, y is a feasible solution for instance x of U_n and compute the cost function $\mathrm{cost}_n(x, y)$.

(2) Every approximation algorithm $\mathcal{A}$ for problem $U_{R,n}$ *answering* $x_1, \ldots, x_j$ *correct* has approximation ratio $A.R.(\mathcal{A}) \leq 1 + 2^{-j}$.

Every approximation algorithm $\mathcal{A}$ for problem $U_{R,n}$ *answering* x_j *incorrect* has approximation ratio $A.R.(\mathcal{A}) \geq 1 + 2^{-(j+2)}$.

(3) For $j = 1, \ldots, n-1$, there exists a randomized approximation algorithm $\mathcal{A}$ with running time

$$O\left(\sum_{i=1}^{j} |X|^{2^i}\right) \quad = \quad O\left(j \cdot |X|^{2^j}\right)$$

such that $\Pr\{\mathcal{A}$ on input x answers $x_1, \ldots, x_j$ correct$\} \geq \frac{3}{4}$

(4) For $j = 1, \ldots, n-1$: There is no randomized approximation algorithm with running time $O\left(|X|^{2^{j-1}}\right)$ for $U_{R,n}$ answering x_j correct.

(5) There is no randomized approximation algorithm with polynomial running time for $U_{R,n}$ answering x_n correct.

The proof of (1) and (2) is identical to the deterministic case. Let $\mathcal{A}$ be a $T(n)$-time bounded randomized algorithm for the functional problem Π_R such that

$$\Pr\{(x, \mathcal{A}(x)) \in R\} \geq \frac{3}{4}$$

for every $x \in L_R$. By the δ-Lemma for Monte-Carlo algorithms, there is a randomized algorithm $\mathcal{A}'$ such that

$$\Pr\{x \in L_R \Rightarrow (x, \mathcal{A}'(x)) \in R \text{ in time bounded by } f(j) \cdot T(|x|)\} \geq \left(\frac{3}{4}\right)^{1/j}$$

(for some function $f(j)$ of j). Let $\mathcal{B}$ be the approximation algorithm for U_n which for each component x_j independently sets $\pi_j := \mathcal{A}'(x_j)$.

We obtain

$$\begin{aligned} & \Pr\{\mathcal{B} \text{ on input } X \text{ answers } x_1, \ldots, x_j \text{ correct}\} \\ = \; & \prod_{i=1}^{j} \Pr\{x_i \in L_R \Rightarrow (x_i, \mathcal{A}'(x_i)) \in R\} \\ \geq \; & \left(\frac{3}{4}\right)^{j/j} = \frac{3}{4} \end{aligned}$$

which completes the proof of (3).

In order to prove (4) we make use of the same kind of reduction from Π_R to $U_{R,n}$ as before: Given an instance x of Π_R we construct an instance X of $U_{R,n}$ such that $X = (x_1, \ldots, x_n, 0^k)$, $\quad x = x_j$ and

$$|X| \in \left[\sqrt[2^j]{T(|x|)}, \ \sqrt[2^j]{T(|x|)} + 1 \right]$$

Now assume $\mathcal{B}$ is a randomized approximation algorithm for $U_{R,n}$ such that

$$\Pr\left\{\mathcal{B} \text{ answers } x_j \text{ correct in time } c \cdot \left(|X|^{2^{j-1}}\right)\right\} \geq \frac{3}{4}$$

for some $c > 0$. Since

$$c \cdot |X|^{2^{j-1}} \leq c \cdot \left(\sqrt[2^j]{T(|x|)} + 1\right)^{2^{j-1}} = O\left(2^{j-1} \cdot T(|x|)^{\frac{2^{j-1}}{2^j}}\right) = O\left(2^{j-1} \cdot t(|x|)\right)$$

this contradicts assumption (A”), hence (4) holds as well.

Let $(M_i, c_i), i \in \mathbb{N}$ be a listing of all pairs consisting of PTMs M_i and constants $c_i > 0$, where we initially guess M_i being an approximation scheme for problem U_R with error probability bounded by 1/4. Assume that each pair occurs infinitely often in this list. For each such M_i, either the guess is incorrect and we will recognize this after finite amount of time, or we diagonalize against M_i being an efficient randomized approximation scheme for U_R with error probability bounded by $\frac{1}{4}$.

We construct problem U_R in stages. In stage n the n-th interval $I_n = [x_{n-1}, x_n)$ is defined (recall that this refers to lexicographic order on Σ^*), U_R restricted to I_n will be defined as $U_{R,n}$. During the construction the following requirements are satisfied:

(C_n) For $j = 1, \ldots, n, i = 1, \ldots, n$:
There are strings $y = y_{i,j}$ and $x = x_i$ in $I_n = [x_{n-1}, x_n)$ such that

$$Pr\left\{\text{A.R. of } M_i(y, 2^{j+3}) \text{ is } \leq 1 + \tfrac{1}{2^{j+3}} \text{ within time } c_i \cdot |y|^{2^{j-1}}\right\} < \tfrac{3}{4}$$
$$Pr\left\{\text{A.R. of } M_i(x, 2^{n+3}) \ \leq 1 + 2^{-(n+3)} \text{ within time } c_i \cdot |x|^{2^n}\right\} < \tfrac{3}{4}$$

Let us argue that (assuming we guarantee problem U_R being in $RPTAS$) constraints (C_n) are sufficient in order to prove the theorem: Hence assume $U_R \in REPTAS$ but (C_n) are satisfied. Then due to the fact that we can decrease error probability to $\frac{1}{4}$ increasing running time by at most a constant factor, there exists an efficient randomized polytime approximation scheme

M for problem U_R with error probability bounded by $\frac{1}{4}$ and randomized running time bounded by

$$f(n) \cdot |x|^\alpha \text{ for some } \alpha > 0 \text{ and some function } f(n).$$

Now choose j such that $\alpha < 2^{j-1}$, and let i be choosen such that $(M_i, c_i) = (M, f(2^{j+3}))$. From some stage n on (namely $n \geq \max\{i, \log(\alpha)\}$) there exist strings y in interval I_n such that

$$Pr\left\{\text{A.R. of } M_i(y, 2^{j+3}) \text{ is} \leq 1 + \tfrac{1}{2^{j+3}} \text{ within time } f(2^{j+3}) \cdot |x|^\alpha\right\} < \frac{3}{4},$$

hence we obtain a contradiction and thus M is not an efficient randomized polytime approximation scheme for problem U_R.

Construction of Problem U_R in Stages.

Stage 0: Set $x_0 := 0$.

Stage $n > 0$: Let x_{n-1} and U_R restricted to the interval $[x_0, x_{n-1})$ already be defined. We will construct x_n and define

$$U_R \quad \text{restricted to interval } I_n \text{ be equal to } U_{R,n}.$$

Now by brute force find strings $y_{i,j}, x_i, 1 \leq i, j \leq n$ satisfying constraint (C_n). Such strings are guaranteed to exist due to properties (1)-(5) of problem $U_{R,n}$. Let T_n be the total time needed to (deterministically) construct problem U_R up to stage $n-1$ and to find strings $y_{i,j}, x_i, 1 \leq i, j \leq n$ Let $x_n := 0^{T_n}$.
End of Stage n

By the very same reason as in the construction in the proof of theorem 12 in section 4.3, problem U_R has polynomial time decidable sets of instances and solutions and polynomial cost functions, furthermore we obtain a randomized approximation scheme for U_R by first computing the interval number of string x, if it is too small solve the instance to optimality by brute force and otherwise answering sufficiently many components of instance x corrcectly by the randomized algorithm for Π_R. Hence $U_R \in RPTAS \setminus REPTAS$, which completes the proof. □

7 Conclusion

We have proved a separation of the classes $PTAS$ and $EPTAS$ under the assumption that there are NP search problems with a superpolynomial lower bound on the deterministic time complexity. We have shown that there exists no relativizing proof that our assumption implies $FPT \neq W[P]$, hence in this sense our result can be seen as independent from the separation result given by Cesati and Trevisan. Furthermore we obtain similar results for the case of randomized polynomial-time approximation schemes. It remains as an open problem to prove the strictness of the inclusion $EPTAS \subseteq PTAS$ under assumption $P \neq NP$.

Acknowledgement. We would like to thank Marek Karpinski and Claus Viehmann for valuable discussions.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. Complexity and Approximation. Springer, 1999

[2] C. Bazgan. Schemas d'approximation et complexite' parametree. *Thesis, INRIA, Orsay, France*, 1995.

[3] Richard Beigel and Judy Goldsmith. Downward separation fails catastrophically for limited nondeterminism classes. In *Structure in Complexity Theory Conference*, pages 134–138, 1994.

[4] L. Cai and J. Chen. On fixed-parameter tractability and approximability of np optimization problems. *Journal of Computer and System Sciences*, 54(3):465–474, 1997.

[5] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(47):165–171, 1997.

[6] Y. Chen and M. Grohe. An Isomorphism Between Subexponential and Parameterized Complexity Theory. *Proceedings of the 21st IEEE Conference on Computational Complexity (CCC'06)*, PP.314-328, 2006.

[7] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. In *Congr. Num. 87*, pages 161–187, 1992.

[8] R. G. Downey and M. R. Fellows. Parameterized Complexity. Springer, 1997

[9] J. Flum, M. Grohe and M. Weyer. Bounded fixed-parameter tractability and log2n nondeterministic bits , *Journal of Computer and System Sciences* 72:34-71, 2006.

[10] J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006

[11] M. Hauptmann. *Approximation Complexity of Optimization Problems: Structural Foundations and Steiner Tree Problems*, PhD-Thesis, University of Bonn, 2004, URL: `http://hss.ulb.uni-bonn.de/diss_online/math_nat_fak/2004/hauptmann_mathias/0380.pdf`.

[12] M. Hauptmann. *The Measure Hypothesis and Efficiency of Polynomial Time Approximation Schemes*, Proc. Tenth Italian Conference on Theoretical Computer Science, p. 151–162, World Scientific, 2007.

[13] J. M. Hitchcock. Small spans in scaled dimension. *SIAM Journal on Computing*, 34(1):170–194, 2004.

[14] J. M. Hitchcock, J. H. Lutz, and E. Mayordomo. Scaled dimension and nonuniform complexity. *Journal of Computer and System Sciences*, 69:97–122, 2004.

[15] J. M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 336–347, 2004.

[16] C.M.R. Kintala and P. Fischer. Refining nondeterminism in relativized complexity classes. *SIAM Journal on Computing*, 13:329–337, 1984.

[17] J. H. Lutz. Category and measure in complexity classes. *SIAM Journal on Computing*, 19:1100–1131, 1990.

[18] J. H. Lutz. Dimension in complexity classes. In *IEEE Conference on Computational Complexity*, pages 158–169, 2000.