Electronic Colloquium on Computational Complexity, Report No. 98 (2004)

Promise Hierarchies

Lance Fortnow* Rahul Santhanam* Luca Trevisan†

Abstract

We show that for any constant a, $\mathsf{ZPP}/b(n)$ strictly contains $\mathsf{ZPTIME}(n^a)/b(n)$ for some $b(n) = O(\log n \log\log n)$. Our techniques are very general and give the same hierarchy for all the common promise time classes including RTIME, $\mathsf{NTIME} \cap \mathsf{coNTIME}$, UTIME, MATIME, AMTIME and BQTIME.

We show a stronger hierarchy for RTIME: For every constant c, $\mathsf{RP}/1$ is not contained in $\mathsf{RTIME}(n^c)/(\log n)^{1/2c}$. To prove this result we first prove a similar statement for NP by building on Žák's proof of the nondeterministic time hierarchy.

1 Introduction

Hartmanis and Stearns [HS65], in their seminal paper on computational complexity, showed that given more time one can compute more languages. For example there exist languages computable in deterministic n^5 time not computable in time $O(n^2)$.

Hartmanis and Stearns used a simulation and diagonalization style proof, a technique we cannot directly use on all complexity classes. For nondeterministic time, we cannot complement and use diagonalization directly. Cook [Coo72] showed how to get around this problem by using padding to create large enough collapses to diagonalize.

Now consider a class like BPP where we do have complementation. The Hartmanis-Stearns approach still fails to work because we cannot directly simulate. BPP is an example of a promise class, where given a probabilistic polynomial-time machine we require that the accepting probability is either high or low for every input. It is undecidable to determine whether a given machine fulfills this promise and straightforward simulation will result in a machine that itself does not fulfill the promise.

Time hierarchies for promise classes remain open but recently Barak [Bar02] and Fortnow and Santhanam [FS04] have shown a time hierarchy for BPTIME if we allow some nonuniform advice. They show that BPP with one-bit of advice is not contained in $\mathsf{BPTIME}(n^c)/1$. However their proofs use specific properties of BPP and does not generalize well to other promise classes.

In this paper we give a very general time hierarchy theorem for promise classes. For example, we show that for any constant a, $\mathsf{ZPP}/b(n)$ strictly contains $\mathsf{ZPTIME}(n^a)/b(n)$ for some $b(n) = O(\log(n)\log(\log(n))$, where ZPP is zero-error probabilistic expected polynomial-time. We can also get a hierarchy for one bit of advice if we use quasipolynomial time. Our techniques apply to every natural promise measure including $\mathsf{NTIME} \cap \mathsf{coNTIME}$, AMTIME (Arthur-Merlin games with time-bounded Arthur), MATIME (Merlin-Arthur games with time-bounded Arthur), UTIME (NTIME with unambiguous accepting paths) and BQTIME (bounded-error quantum). Note our techniques do not require classes closed under complementation.

*University of Chicago, {fortnow,rahul}@cs.uchicago.edu

†University of California at Berkeley, luca@eecs.berkeley.edu

ISSN 1433-8092

We show a stronger hierarchy for one-sided randomized time. The class $\mathsf{RP}/1$ is not contained in $\mathsf{RTIME}(n^c)/(\log n)^{1/2c}$ for any constant c. To prove this result we show the following result for nondeterministic time: NP is not contained in $\mathsf{NTIME}(n^c)/(\log n)^{1/2c}$ for any constant c. This is the first nontrivial hierarchy against nonuniform nondeterministic time.

1.1 Our Results

Let CTIME be RTIME, ZPTIME, $\mathsf{NTIME} \cap \mathsf{coNTIME}$, AMTIME, MATIME, UTIME, BQTIME or a similar class.

Theorem 1. *For every constant $\alpha > 1$, there exists a constant $\gamma > \alpha$ and an advice bound $b(n) = O(\log(n) \log\log n)$ such that*

$$\mathsf{CTIME}(n^\alpha)/b(n) \subsetneq \mathsf{CTIME}(n^\gamma)/b(n).$$

Theorem 2. *For every constant $\alpha > 1$, there exists a constant $\gamma > \alpha$ such that*

$$\mathsf{CTIME}(2^{(\log(n))^\alpha})/1 \subsetneq \mathsf{CTIME}(2^{(\log(n))^\gamma})/1.$$

Theorem 3. *For every constant $\alpha > 1$, there is a constant $\alpha' > \alpha$ such that*

$$\mathsf{RTIME}(n^{\alpha'})/1 \not\subseteq \mathsf{RTIME}(n^\alpha)/(\log n)^{1/2\alpha}.$$

2 Preliminaries

We assume a basic familiarity with complexity classes [BDG88, BDG90]. At various points in our paper, we will use a standard easily computable pairing function $<>$: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that $| < x, y > | \leqslant |x| + |y| + O(\log |x|)$.

We make the distinction between *syntactic* and *promise* (also called semantic [Pap94]) time-bounded complexity classes . The distinction is perhaps best explained through an example. For any constructible time bound t, the class $\mathsf{DTIME}(t)$ is a syntactic class, in that there is an effective enumeration of deterministic time t machines and each machine in this enumeration defines a language in $\mathsf{DTIME}(t)$. $\mathsf{NTIME}(t)$ is another example - here, the enumeration is of time t non-deterministic machines. In fact, effective enumerability is equivalent to the existence of *complete* problems for these classes. The proofs of hierarchy theorems for these classes use diagonalization, which relies critically on the effective enumerability property.

There are also natural classes which are not known to have an effective enumeration of machines defining the class. An example of such a class is $\mathsf{BPTIME}(t)$, for any time bound t. There is a natural enumeration of machines underlying the class BPTIME, namely the enumeration of probabilistic machines halting in time t, however it is not true that all machines in this enumeration define languages in $\mathsf{BPTIME}(t)$. For instance, there are machines in this enumeration which make error arbitrarily close to $1/2$ on their inputs, which violates the bounded-error property required of BPTIME machines. It is undecidable to check whether a probabilistic machine is a BPTIME machine or not, since the property "M is a BPTIME machine" involves a universal quantification over all inputs x of the property "M is a BPTIME on input x" (which is satisfied if and only if M has probability at least $2/3$ of accepting x, or probability at least $2/3$ of rejecting x). We think of the condition "M is a BPTIME machine on input x" as a *promise* condition, which M must satisfy on each input in order to define a language in $\mathsf{BPTIME}(t)$. Thus the key difference between syntactic and promise classes is this: for a syntactic class, each machine in the enumeration defines a language

in the class, while for a promise class, only those machines satisfying the promise condition on all inputs define a language in the class.

Several interesting classes apart from BPP are promise classes, including RP, NP $\cap$ coNP, UP, AM, MA and ZPP. Since promise classes are not known in general to have an effective enumeration, we cannot use diagonalization to show hierarchy theorems for these classes. In this paper, we show hierarchy theorems for general promise classes with small advice. To prove our theorems in full generality, we would need to give a formal characterization of a promise class and use only elements of that characterization in our proofs. We can use this approach by defining the properties required of a promise class which include efficient versions of the universality, an s-m-n theorem for the underlying enumeration, closure of the underlying enumeration under deterministic m-reductions, and that the promise class with time bound t lies between deterministic time t and deterministic time 2^t. These are all properties that can be seen to hold for any of the classes we mentioned earlier, as well as for a number of other classes studied in the literature. However, in the interests of readability, in this extended abstract we give a more informal presentation. All the steps in our proofs may be observed to go through for any reasonable promise class.

Let $\mathbb{M}$ be an enumeration of Turing machines. We take as our primitive the notion "M is a CTIME(t)-machine on input x", for an $M \in \mathbb{M}$. This should be interpreted to mean that M operates in time $t(|x|)$ and satisfies the promise corresponding to class CTIME on input x. CTIME(t) is the class of languages L such that $L = L(M)$ for a CTIME(t)-machine M.

As an example, for the class ZPTIME(t), the underlying enumeration consists of probabilistic machines outputting "1", "0" or "?" on each computation path, M is a ZPTIME(t)-machine on input x if all computation paths of M on input x halt in time t and either M outputs "1" with probability at least 1/2 and "?" otherwise (in which case we say $M(x) = 1$) or M outputs "0" with probability at least 1/2 and "?" otherwise (in which case we say $M(x) = 0$). If for any input x, $M(x)$ has both an accepting or rejecting path or the probability that $M(x)$ outputs "?" is greater than 1/2 then M does not fulfill the ZPTIME promise and is not a ZPTIME machine.

We need to define a notion of promise classes with advice. $L \in$ CTIME(t)/$a(n)$ if there is an advice function $s : N \to \{0,1\}^{a(n)}$ and a machine $M \in \mathbb{M}$ such that for each n, for all x of length n, M is a CTIME(t)-machine on input $< s(|x|), x >$, and $x \in L$ iff $M(< s(|x|), x >) = 1$. Note that in the above definition M is not required to be a CTIME(t)-machine on input $< y, x >$ where $y \neq s(|x|)$ which differs from the original definition of advice of Karp and Lipton [KL82]. Proving a hierarchy theorem with advice in the Karp-Lipton model would imply a hierarchy theorem without advice. For syntactic classes, the two definitions are equivalent, and hence proving a hierarchy theorem with advice is harder than proving one without advice. We note that since syntactic classes are special cases of promise classes (with a trivial promise), our hierarchy theorems go through for them as well.

3 Our Techniques

Until recently we knew essentially no techniques for proving hierarchy theorems for promise classes. In 2002, Boaz Barak [Bar02] had the ingenious idea of using *optimal algorithms* for proving hierarchy theorems. Using his idea and extensions thereof, hierarchy theorems have been shown for BPP with 1 bit of advice [Bar02, FS04] and for average-case BPP [FS04]. A natural question is whether this technique generalizes to prove hierarchy theorems for other promise classes such as ZPP, NP $\cap$ coNP and AM.

We briefly review Barak's technique. An optimal algorithm A for a language L is an algorithm which is at most polynomially slower than *any* algorithm for L. Barak showed that any EXP-

complete language L has an optimal algorithm, using results from the theory of probabilistically checkable proofs. If the optimal algorithm takes time T, there is a constant $\varepsilon > 0$ such that $L \in \mathsf{BPTIME}(T) - \mathsf{BPTIME}(T^\varepsilon)$, and one can then derive a hierarchy theorem (with advice) using a translation argument.

We do not know how to apply Barak's optimal algorithm technique to general promise classes. In this paper we give a general method to derive hierarchy theorems for promise classes with advice. The key observation is that if we are aiming for a hierarchy theorem for advice, it is sufficient to have *nonuniform* optimal algorithms, i.e., we can use a different optimal algorithm for each input length. It turns out that such algorithms exist for promise classes under very general conditions. Essentially given a complexity class CTIME we take a language L that is known to be hard for deterministic time and try to speed up CTIME decidability of L by using a small additional amount of advice. If we don't succeed, then there is an $\varepsilon > 0$ such that L is decidable in $\mathsf{CTIME}(t)$ but not in $\mathsf{CTIME}(t^\varepsilon)$ on inputs of length n. We can now attempt to use a translation argument to get a hierarchy theorem with advice. If we do succeed, then we try to speed up the computation even further using some more advice. The speedups cannot continually succeed because that would imply that the language L we are trying to decide isn't hard. Much of the work in the proof goes towards reducing the amount of advice required. We end up with a hierarchy theorem for poly-time promise classes with $O(\log(n)\log(\log(n)))$ bits of advice. We can reduce the advice to just one bit at the cost of only obtaining a hierarchy for quasi-polynomial time rather than polynomial time.

We also consider the question of whether there are specific promise classes for which we can do better than in the general result, as in the case of BPP. We manage to show a hierarchy theorem for RP with 1 bit of advice. Since RP doesn't seem to have nice closure properties (in particular, it is a one-sided class), it isn't immediately clear how to define a language with an optimal RTIME algorithm. We show that such an algorithm can be defined for the satisfiability problem by using Levin's idea of an optimal search algorithm. If $\mathsf{NP} \neq \mathsf{RP}$, then the optimal algorithm shows that SAT is in $\mathsf{RTIME}(t(n))$ but not, say, in $\mathsf{RTIME}(t(n)^{1/3})/\log t(n)^{1/3}$ for some time bound $t(n)$, and a padded version of SAT proves the hierarchy theorem. To deal with the case $\mathsf{RP} = \mathsf{NP}$, we prove a new hierarchy theorem for NP against $o(\log(n))$ bits of advice, which may be of independent interest, and we derive the hierarchy theorem for non-uniform RTIME from our hierarchy theorem for non-uniform NTIME.

4 General Promise Classes

4.1 Polynomial Time-Bounded Classes with Small Advice

Let $\mathsf{CTIME}(\cdot)$ be any time-bounded promise class and $\mathbb{M}$ be the underlying enumeration for the class. We prove the following theorem, which implies Theorem 1.

Theorem 4. *For every constant $\alpha > 1$, there exists a constant $\gamma > \alpha$ and an advice bound $b(n) = O(\log(n)\log(\log(n)))$ such that $\mathsf{CTIME}(n^\gamma)/b(n) \not\subseteq \mathsf{CTIME}(n^\alpha)/(b(n)+1)$.*

We require the following diagonalization lemma in our proof of Theorem 4:

Lemma 5. *For any constants k and k' such that $k > k' > 0$, $\mathsf{DTIME}(2^{n^k}) \not\subseteq i.o.\mathsf{DTIME}(2^{n^{k'}})/(n - \log(n))$.*

Proof. The proof is a standard diagonalization. Let $\{M\}_i$ be a recursive enumeration of advice-taking deterministic Turing machines running in time $2^{n^{k'}}$ on inputs of length n. We construct a deterministic Turing machine M running in time 2^{n^k} such that $L(M) \notin i.o.\mathsf{DTIME}(2^{n^{k'}})/n - \log(n)$.

Visualize the set of inputs $\{0,1\}^n$ as divided up into n equal-sized "regions" $C_i, i = 1 \ldots n$, where region C_i contains all inputs with rank between $(i-1)2^n/n$ and $i2^n/n$ in the lexicographic ordering of inputs. Given an input x, M first determines the region C_i in which x lies. It then computes j such that x has rank j in the lexicographic ordering of inputs which lie in region C_i. M runs M_i on x with the jth advice string of length $n - \log(n)$. If M_i rejects, then M accepts, otherwise it rejects. There is only a polynomial overhead for M to run M_i over the running time of M_i on input x, thus M can be made to run in time 2^{n^k} for any $k > k'$. As n increases, M diagonalizes against arbitrarily long programs running in time $2^{n^{k'}}$ and taking advice of length $n - \log(n)$, therefore $L(M) \notin i.o.DTIME(2^{n^{k'}})/(n - \log(n))$. □

Before giving the proof of Theorem 4, we sketch the idea behind the proof. We define an explicit language L' such that L' is in $\mathsf{CTIME}(n^\gamma)/b - \mathsf{CTIME}(n^\alpha)/b$. L' is a padded version of the language L defined in the proof of Lemma 5. Let $k' = 3\alpha'$ in the statement of Lemma 5 and $k = \beta$, where $\beta > 3\alpha'$ is any constant, and α' is to be determined later. Since $\mathsf{DTIME}(t) \subseteq \mathsf{CTIME}(t)$ for any constructible t (this is one of the properties we stated for time-bounded promise classes), there is a program deciding $L(M)$ in $\mathsf{CTIME}(2^{n^\beta})$. The idea is that for each input length n, there is some t such that there is a "small" program deciding $L(M)$ correctly on inputs of length n in $\mathsf{CTIME}(t)$ and no "slightly larger" program decides $L(M)$ correctly on inputs of length n in $\mathsf{CTIME}(\sqrt{t})$. If this were not the case, there would be a sublinear size program deciding $L(M)$ on inputs of length n in $\mathsf{CTIME}(n^\alpha)$ and hence in $\mathsf{DTIME}(2^{n^\alpha})$ infinitely often, which is in contradiction to Lemma 5.

The preceding is rather imprecise. Also, there are subtleties and details that haven't been mentioned. We now proceed to give a more detailed argument.

Let L be a language in $\mathsf{DTIME}(2^{n^\beta}) - i.o.\mathsf{DTIME}(2^{n^{3\alpha'}})/(n - \log(n))$, where $\beta > 3\alpha' > 0$, where α' is a constant to be decided later. By Lemma 5, such a language L exists. Let c be the length of a shortest program deciding L in $\mathsf{CTIME}(2^{n^\beta})$. For any integer $t > 1$, let $g(t)$ be the minimum $i \geqslant 0$ such that $t^{2^i} > 2^{n^\beta}$. Observe that for any $t \geqslant 2$, $g(t) \leqslant \beta \log(n)$.

For any $n > 0$, let $h(n)$ be the largest $t \leqslant 2^{n^\beta}$ such that there is a program of length $c + 2\log(\log(n))g(t)$ deciding L correctly in time $\mathsf{CTIME}(t)$ on all inputs of length $\leqslant n$ and there is no program of length $< c + 2\log(\log(n))(g(t) + 2)$ deciding L correctly in time $\mathsf{CTIME}(\sqrt{t})$ on all inputs of length $\leqslant n$. Since $L \notin i.o.\mathsf{DTIME}(2^{n^{3\alpha'}})/n - \log(n)$, $h(n) > n^{2\alpha'}$ for all but finitely many n.

We now define a language L' as follows:

$$L' = \{x1^y \mid y = 2^{2^i} \text{ for some integer } i, y \geqslant |x|, y+ \mid x \mid \geqslant h(|x|)^{1/(2\alpha')}, x \in L\}$$

We need a bit of notation. For any integer m, there is at most one way to write m as $n + y$, where $y \geqslant n$ and $y = 2^{2^i}$ for some integer i. If there is such a way, we let n_m denote the unique n corresponding to m.

Let M_u be a universal machine for the enumeration $\mathbb{M}$. Let e be a constant such that for any program P operating in time t, M simulates P on x in time $max(\mid p \mid^e, t^e)$. We first argue that L' is decidable in $\mathsf{CTIME}(m^{2\alpha' e})$ with advice of length $b(m) = 1 + c + 2\log(\log(n_m))g(h(n_m)) = O(\log(\log(m))\log(m))$, and then argue that L' is not decidable in $\mathsf{CTIME}(m^{\alpha'/e})$ with advice of length $b(m) + 1$.

Lemma 6. $L' \in \mathsf{CTIME}(m^{2\alpha' e})/b(m)$.

Proof. We define an advice-taking machine M' that decides L' correctly on all inputs within the stated bounds. Given an input x' of length m, M' first checks that there is an integer n_m

corresponding to m and that x' is of the form $x1^y$ for some x such that $|x| = n_m$. If not, M' rejects. Next M' uses the first bit of its advice string to decide whether the pad length is sufficient, i.e., if $m \geqslant h(|x|)^{1/(2\alpha')}$. If the advice bit is 0, M' rejects. Observe that all the tests used by M' thus far can be implemented in quasilinear time.

If the advice bit is 1, M' runs M_u on x with advice p, where p is the advice string with the first bit removed. The correct advice p decides L correctly on all inputs of length $\leqslant n_m$ within time $h(n_m)$ hence M' decides L' correctly on all inputs of length m within time $h(n_m)^e \leqslant m^{2\alpha' e}$, i.e., it decides L' correctly and operates within time $m^{2\alpha' e}$, where m is the length of its input. □

Lemma 7. $L' \notin \mathsf{CTIME}(m^{\alpha'/2e})/(b(m)+1)$.

Proof. We will assume $L' \in \mathsf{CTIME}(m^{\alpha'/e})/(b(m)+1)$ and then derive a contradiction. Let M' be a machine deciding L' correctly in $\mathsf{CTIME}(m^{\alpha'/e})/(b(m)+1)$. We show that for almost every n, there is a program P of length $< c + 2(\log(\log(n))(g(h(n))+1)$ deciding L correctly on inputs of length n in $\mathsf{CTIME}(\sqrt{h(n)})$, which is a contradiction to the definition of $h(n)$.

The program P has the minimum i such that $m(n) = n+2^{2^i} \geqslant h(n)^{1/(2\alpha')}$ and the correct advice a to M' for input length $m(n)$ hardwired into it. It requires at most $\log(\log(n))$ bits to specify i, and hence at most $\log(\log(n))+b(m(n))+o(\log(\log(n)))$ bits to specify i, M' and the advice for M'. Since $b(m(n)) = 1+c+2\log(\log(n))g(h(n))$, for n large enough $|P| < c+2(\log(\log(n))(g(h(n))+1)$, as required.

Now we describe how P operates. Given i and a, P pads its input x from length n to length $m(n)$ and runs M' on the padded input with advice a. From the definition of $m(n)$, we have $m(n) \leqslant h(n)^{1/\alpha'}$. Taking into account the overhead in running M', the program P halts in time $\sqrt{h(n)}$ on inputs of length n and decides L correctly, which is a contradiction. □

Now setting $\alpha' = 2\alpha e$ and $\gamma = \alpha e^2$, Theorem 4 follows from Lemma 6 and Lemma 7. A nice feature of our technique is that we can also get unconditional hierarchy theorems with advice even for time t which is not polynomially bounded. The proofs of the hierarchy theorems for BPTIME with advice [Bar02, FS04] do not seem to give any unconditional results for general t.

Corollary 8. *For any time bound t such that $n \leqslant t \leqslant 2^n$, there is a constant $\varepsilon > 0$ and an advice bound $b(n) = O(\log(t)\log(\log(t)))$ such that $\mathsf{CTIME}(t)/b(n) \not\subseteq \mathsf{CTIME}(t^\varepsilon)/(b(n)+1)$*

4.2 Promise Classes with 1 Bit of Advice

Theorem 1 isn't as efficient as we might hope with respect to advice. In this section, we show that if we are willing to settle for slightly weaker hierarchies, namely hierarchies for quasi-polynomial time rather than polynomial time, we are able to reduce the advice required to 1 bit. In order to do this, we apply an idea of Goldreich, Sudan and Trevisan [GST04] of coding the advice string in the length of the pad, after which we need just 1 bit to tell us whether the pad length is large enough and whether the correct advice string has been encoded.

Before applying the advice reduction, we need a variant of Theorem 1 for quasipolynomial time bounds:

Theorem 9. *Let CTIME be a time-bounded promise class. Given constants $\alpha' > 1$ there is a constant $\gamma > \alpha'$ and an advice bound $a(n) \leqslant (\log\log(n))^2$ such that $\mathsf{CTIME}(2^{(\log(n))^\gamma})/a(n) \not\subseteq \mathsf{CTIME}(2^{(\log(n))^{\alpha'}})/(a(n)+1)$.*

Theorem 9 can be obtained using the same proof technique used to show Theorem 1, by tweaking the parameters a little. In the proof of Theorem 1, we used the fact that either there is some t such that there is a simulation of time t algorithms for a hard language $L \in \mathsf{EXP}$ by time $\sqrt{t}$ algorithms with small advice, or we can use padding to derive a hierarchy theorem with advice. To prove Theorem 9, we ask for each t whether there is a more drastic speedup of L, namely a simulation of time $t^{\log t}$ algorithms by time t algorithms with small advice. If not, we get slightly weaker hierarchy theorems than before, but with the advantage that the advice used in the hierarchy is bounded by a function that grows more slowly than the logarithm of the time bound. This feature is essential for us to apply the advice-reduction lemma, which we state below.

Lemma 10. *Let $f : N \to N$ be a function such that $g(n) = \sum_{i=1}^{i=n} 2^{f(i)}$ is time-constructible. For each m, $g^{-1}(m)$ denote the maximum n such that $g(n) \leqslant m$. If there is an advice bound $a(n) \leqslant f(n)$ such that $\mathsf{CTIME}(t_1(n))/a(n) \not\subseteq \mathsf{CTIME}(t_2(n))/(a(n)+1)$, then $\mathsf{CTIME}(t_2(g^{-1}(n))/1 \subsetneq \mathsf{CTIME}(t_1(g^{-1}(n)))/1$.*

Setting $f(n) = (\log\log(n))^2$, $t_1 = 2^{\log n^\gamma}$ and $t_2 = 2^{\log n^{\alpha'}}$ for any $\alpha' > \alpha$ in Lemma 10, and applying the advice reduction to the hierarchy theorem in Theorem 9, we obtain Theorem 2.

5 Hierarchy for Randomized Polynomial Time

In this section we prove Theorem 3, a hierarchy for randomized time with one-bit of advice. Let us first give a formal definition of RTIME with advice.

Definition 11. *$L \in \mathsf{RTIME}(t)/a$ if there is a probabilistic machine M and an advice function $s : \mathbb{N} \to \{0,1\}^{a(n)}$ such that for each n, for all x of length n, M on $< s(|x|), x >$ halts within time $t(n)$ and either accepts with probability at least 1/2 or rejects with probability 1.*

We first need to prove a theorem about nondeterministic time.

Theorem 12. *For each constant $c > 1$, $\mathsf{NP} \not\subseteq \mathsf{NTIME}(n^c)/(\log(n))^{1/2c}$.*

We modify the proof of the nondeterministic time hierarchy due to Žák [Ž83].

Proof. Let $M_1, M_2 \ldots$ be an enumeration of nondeterministic Turing machines. We define a nondeterministic machine M that acts as follows on input $w = 1^i 0 1^m 0 1^{2^k} a_1 a_2 \ldots a_{m^c}$:

1. If $k < m^c$ then simulate M_i on input $1^i 0 1^m 0 1^{2^{k+1}} a_1 a_2 \ldots a_{m^c}$ for $|w|^{2c}$ steps using advice a_k where $|a_k| = \log^{1/2c} |w| \leqslant m$.
2. If $k = m^c$ then accept iff M_i on input $1^i 0 1^m 0 a_1 a_2 \ldots a_{m^c}$ rejects which we can do quickly as a function of the current input size.

This machine runs in nondeterministic polynomial time so by assumption can be simulated in time n^c by some machine M_i with some advice bound a. So we have for sufficiently large m,

$$1^i 0 1^m 0 1 a_1 a_2 \ldots a_{m^c} \in L(M) \Leftrightarrow 1^i 0 1^m 0 1^2 a_1 a_2 \ldots a_{m^c} \in L(M) \Leftrightarrow \cdots$$

$$\Leftrightarrow 1^i 0 1^m 0 1^{2^{m^c}} a_1 a_2 \ldots a_{m^c} \in L(M) \Leftrightarrow 1^i 0 1^m 0 1 a_1 a_2 \ldots a_{m^c} \notin L(M)$$

a contradiction. □

Let us fix an enumeration $M_1, M_2, \ldots$, of probabilistic Turing machine. We assume that if a machine M has a k-bit description, then M occurs within the first $O(2^k)$ places of the enumeration.

```
Algorithm Lev
  Input: φ
  i := 1
  while (true)
    Simulate M_1, ..., M_i on input φ for i steps
    If one machine writes a satisfying assignment for φ on its tape
      halt and accept
    i := 2 * i
```

Figure 1: Levin's optimal search algorithm.

In the interest of generality, we do not fix a specific model of computation, but we just state a few axioms that we assume our machine model to satisfy. We assume that we are using a model of computation such that given a machine M that can be described using k bits, given a string x and a time bound t, it is possible to simulate t steps of the computation of $M(x)$ in time $O(t^{1+o(1)} \cdot k^{O(1)})$. This assumption is satisfied, for example, by c-tape Turing machines for every $c > 1$. Finally, we assume that there is an $O(n^2)$ algorithm to check if a given assignment satisfies a given 3SAT formula, where n is the bit length of a description of the formula.

Levin [Lev73] proposed a "fastest search algorithm" for search problems in NP. On input, say, a 3SAT formula, the algorithm would enumerate all possible algorithms and then simulate them, in a certain order, for increasing running times, until one of them finds a satisfying assignment for the formula. If the formula is unsatisfiable, then this process runs for ever, but if the formula is satisfiable then it eventually finds a satisfying assignment, while running at most linearly slower that any other algorithm for 3SAT. In Figure 1 we describe a probabilistic version of the algorithm.

We note that:

- If φ is not satisfiable, then $Lev(\varphi)$ does not halt.
- If φ is satisfiable, and if there is a probability at least p that machine M_i finds a satisfying assignment for φ within t steps, then there is a probability at least p that $Lev(\varphi)$ finds a satisfying assignment within $O((\max\{i,t\})^{2+o(1)} + \max\{i,t\} \cdot n^2)$, where n is the size of φ.

Let us define the worst-case median running time of algorithm Lev as follows. (The worst-case is over all inputs, the median is over the coin tosses of the algorithm.)

Definition 13. *For a satisfiable formula φ, we define $L(\varphi)$ as the smallest t such that $Lev(\varphi)$ finds with probability at least 1/2 a satisfying assignment for φ within t steps.*

For every n, we define $L(n)$ to be the largest value of $L(\varphi)$ over all satisfiable formulas φ of size n.

We note that the definition of $L(\varphi)$ is well posed: it is impossible that $L(\varphi)$ find a satisfying assignment in zero steps, and there is probability one that it will find such an assignment within $2^{O(n)}$ steps. Furthermore, the probability is non-decreasing with the number of steps, so there must a place where the probability goes from being smaller than 1/2 to being at least 1/2.

The "optimality" of the algorithm is given by the following simple result, which is a consequence of previous observations.

Lemma 14. *If 3SAT$\in \mathsf{RTIME}(t(n))/a(n)$, then*

$$L(n) \leqslant O((\max\{2^{a(n)}, t(n)\})^{2+o(1)} + \max\{t(n) + 2^{a(n)}\} \cdot n^2) \ .$$

In order to prove our hierarchy theorem, we will consider two cases. If $L(n)$ is polynomially bounded, then $\mathsf{NP} = \mathsf{RP}$ and a hierarchy theorem for RTIME follows from the hierarchy theorem for NTIME. If $L(n)$ is super-polynomial, then it follows that 3SAT can be solved in $\mathsf{RTIME}(L(n))$ but not in $\mathsf{RTIME}(L(n)^\varepsilon)/\log(L(n))^\varepsilon$ for some constant ε. The hierarchy theorem then follows from a translation argument.

Lemma 15. *Suppose that there is a polynomial p such that, for all n, $L(n) \leqslant p(n)$.*
Then, for every constant c,

$$\mathsf{RP}/1 \not\subseteq \mathsf{RTIME}(n^c)/((\log n)^{1/2c})$$

Proof. Under the assumption of the lemma, Algorithm *Lev* is an RP algorithm for 3SAT, and so $\mathsf{NP} = \mathsf{RP}$, and also $\mathsf{NP}/a(n) = \mathsf{RP}/a(n)$ for every a. Furthermore, by definition, we have $\mathsf{RTIME}(t(n))/a(n) \subseteq \mathsf{NTIME}(t(n))/a(n)$ for every a and t. Suppose towards a contradiction that for some constant c we had

$$\mathsf{RP}/1 \subseteq \mathsf{RTIME}(n^c)/((\log n)^{1/2c}) \ .$$

Then we would have

$$\mathsf{NP}/1 = \mathsf{RP}/1 \subseteq \mathsf{RTIME}(n^c)/((\log n)^{1/2c}) \subseteq \mathsf{NTIME}(n^c)/((\log n)^{1/2c})$$

which contradicts Theorem 12. □

It remains to consider the other case, in which $L(n)$ is superpolynomial. In such a case, we prove the hierarchy theorem using a padded version of 3SAT defined as follows.

Definition 16. *For a constant $c \geqslant 1$, the language D_c contains instances $(\varphi, 1 \cdots 1)$ obtained by padding a 3CNF formula φ with ones. A string $(\varphi, 1 \cdots 1)$ of length m is a YES instance of D_c if, when we write $m = q^2 + r$ with $r < q$ in a unique way, we have that*

- *r equals the length of φ.*
- *$q = \lfloor T^{1/6c} \rfloor$ where T is $L(r)$ rounded down to the nearest power of two,*
- *φ is satisfiable.*

Intuitively D_c contains satisfiable instances φ of 3SAT that have been padded to length approximately $L(\varphi)^{1/3c}$, so that *Lev* can solve them in time roughly m^{3c}, where m is the length of the padded instance, and a much faster algorithm cannot exist otherwise it could be used to violate the optimality of *Lev*.

The complications in the definition of D_c arise from the need to define the padding in such a way that, for a given string $(\varphi, 1 \cdots 1)$, one can efficiently check if the length of the padding is correct. As in [FS04] and [GST04], we use the input length to encode an approximation of L. Specifically, we observe that every integer m can be seen as specifying a pair of integers (r, q), for example by writing $m = q^2 + r$ with $r < q$ in a unique way. Then we only accept as valid the strings $(\varphi, 1 \cdots 1)$ such that the total length m encodes integers (q, r) where r is the length of φ, and q is about $L(r)^{1/6c}$. One bit of advice can tells us whether m has the required property (that it encodes r, q such that $q \approx L(r)^{1/6c}$) and, given this information, the rest can be easily checked.

Lemma 17. *For every constant c*

$$D_c \in \mathsf{RTIME}(O(n^{3c}))/1$$

Proof. Given an input $(\varphi, 1\cdots 1)$ of length n, we write $n = q^2 + r$ with $r < q$ and the one bit of advice tells us whether $q = \lceil T^{1/6c} \rceil$, where T is the $L(r)$ rounded down to the nearest power of two.

If the advice bit tells us that the condition is not satisfied, then we reject. Otherwise, we check that r is the length of φ (otherwise we reject) and then run algorithm Lev on φ. If φ is satisfiable, then there is a probability at least $1/2$ that we find a satisfying assignment and halt within $L(r) \leqslant 2T = \Theta(q^{6c}) = \Theta(n^{3c})$ steps. □

Lemma 18. *Suppose that for some constant c*

$$D_c \in \mathsf{RTIME}(O(n^c))/\log n^c \ .$$

Then

$$3SAT \in \mathsf{RTIME}\left(O(n^{7c} \cdot (L(n))^{1/3})\right) / \log(L(n))^{1/3} + \log\log L(n) + O(1) \ .$$

Proof. Let A be the algorithm for D_c that uses $\log n^c$ bits of advice, never accepts NO instance of D_c and such that every YES instance of D_c of length n has a probability at least $1/2$ of being accepted within $O(n^c)$ steps.

We describe an algorithm for 3SAT whose complexity matches the conclusion of the lemma. On input φ of length n, the advice of the algorithm is a number T that equals $L(n)$ rounded down to the nearest power of two. (It takes only $\log\log L(n)$ bits to specify T.)

If $\lfloor T^{1/6c} \rfloor < n$, then we run $L(\varphi)$ for $2T$ steps and accept if and only L finds a satisfying assignment. The simulation takes time $T^{1+o(1)} \leqslant O(n^{7c})$, and, by the definition of T, if φ is satisfiable then there is a probability at least $1/2$ that we find a satisfying assignment for φ.

If $\lfloor T^{1/6c} \rfloor > n$, then we let $m = \lfloor T^{1/6c} \rfloor^2 + n$ and we construct a padded instance $(\varphi, 1\cdots 1)$ of length m, and we run $A((\varphi, 1\cdots 1))$. In this case, the advice contains also the advice string of length $(\log m^c) \leqslant \log(L(n))^{1/3} + O(1)$ to use to correctly execute A. With the correct advice string, if φ is satisfiable, then A has a probability at least $1/2$ of accepting φ within $O(m^c) = O(L^{1/3})$ steps. □

Lemma 19. *Suppose that for some constant k*

$$3SAT \in \mathsf{RTIME}\left(O(n^k \cdot (L(n))^{1/3})\right) / \log(L(n))^{1/3+o(1)} \ .$$

Then there is a polynomial p such that $L(n) \leqslant p(n)$ for every n.

Proof. The assumption and Lemma 14 implies $L(n) \leqslant O(n^{2k} L(n)^{2/3+o(1)})$ and so $L(n) \leqslant O(n^{6k+o(1)})$. □

We are finally ready to prove Theorem 3.

Proof. [Of Theorem 3] Suppose towards a contradiction that there is a constant α such that

$$\mathsf{RP}/1 \subseteq \mathsf{RTIME}(n^\alpha)/(\log n)^{1/2\alpha}$$

and consider the language D_α. By Lemma 17, we have $D_\alpha \in \mathsf{RP}/1$, and by the above assumption, we would have $D_c \in \mathsf{RTIME}(n^\alpha)/(\log n)^{1/2\alpha}$. Then, Lemmas 18 and 19 would imply that there is a polynomial p such that $L(n) \leqslant p(n)$ for every n, which, using Lemma 15 gives us $\mathsf{RP}/1 \not\subseteq \mathsf{RTIME}(n^\alpha)/(\log n)^{1/2\alpha}$ □

References

[Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for "Slightly Non-uniform" algorithms. *Lecture Notes in Computer Science*, 2483:194–208, 2002.

[BDG88] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity 1*. Springer-Verlag, New York, NY, 1988.

[BDG90] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity 2*. Springer-Verlag, New York, NY, 1990.

[Coo72] Stephen A. Cook. A hierarchy for nondeterministic time complexity. In *Conference Record, Fourth Annual ACM Symposium on Theory of Computing*, pages 187–192, Denver, Colorado, 1–3 May 1972.

[FS04] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, 2004.

[GST04] Oded Goldreich, Madhu Sudan, and Luca Trevisan. Personal Communication, 2004.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. (AMS)*, 117:285–306, 1965.

[KL82] Richard Karp and Richard Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[Lev73] Leonid Levin. Universal search problems(in russian). *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

[Pap94] Christos Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[Ž83] S. Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, October 1983.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'