SIGACT News Complexity Theory Column 52

Lane A. Hemaspaandra
Dept. of Computer Science, University of Rochester
Rochester, NY 14627, USA

Introduction to Complexity Theory Column 52

Bovet and Crescenzi's Complexity Textbook Is Again Available

For those who are big fans of the fantastic complexity textbook by Daniel Bovet and Pierluigi Crescenzi [BC93] (and I myself certainly am), there is great news. The authors have made their book available online, free of charge for noncommercial use. It can be found at via Pilu's web site (start at `http://www.algoritmica.org/piluc`, then click on the "Books" section, and then click on "Introduction to the Theory of Complexity").

Speaking of textbooks, I see (via `www.aw-bc.com/home`) that a third edition of Hopcroft–Motwani–Ullman—which of course in its first edition was Hopcroft–Ullman—has just come out. Nifty!

This Issue's Column

Oh, you nasty tricksy classes with promises! You know who you are: UP, FewP, ZPP, R, BPP, NP $\cap$ coNP, and the other usual suspects. Why can't you be more like your well-behaved sibling NP? NP cleans up her toys and comes in for dinner when called. She has nice enumerations of machines of her own type that cover her exactly, she (not at all unrelatedly) has complete sets, she contains sparse sets not in her brother P exactly if their exponential-time cousins differ, she has (well, nondeterministic time itself has) a very tight time hierarchy theorem, and if she and P are equal with respect to even one tally oracle then they are equal in the real world. Many other promise-free classes share many of these properties. However, you nasty tricksy classes with seemingly nontrivializable promises at your core (e.g., "this machine will have either 0 or 1 accepting paths on all inputs" or "there is another NP machine that accepts exactly the complement of this set"), you often seem to potentially lack many or all of these properties. To learn about your own evil ways, you nasty tricksy classes, you might want to look at some of the existing literature about your stormy nature (e.g., [Sip82, Gur83, HH88, HR92, HJV93, Bor94, Rot95, HJ95]), though you might want also to look at work praising your (infrequent) good behavior [RRW94]. But right now, why don't you look below, nasty tricksy classes, and read the very nice article that Lance Fortnow and Rahul Santhanam have prepared on time hierarchy theorems. (Indeed, nasty tricksy classes, you might even spot your own names and learn how the context of advice may put you on better behavior regarding time hierarchies.) And, nasty tricksy classes, just for your information, the next issue's column will be by Jin-Yi Cai and Osamu Watanabe, and the one in an issue coming out not too long after that one will be by Rolf Niedermeier; so stay tuned!

Guest Column: Recent Work on Hierarchies for Semantic Classes[1]

Lance Fortnow[2]
Rahul Santhanam[3]

Abstract

We survey time hierarchies, with an emphasis on recent work on hierarchies for semantic classes.

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 similar to those used by Cantor [Can15] and Turing [Tur36]. This method has the machine with the larger time bound simulate machines with smaller time bounds and then negate the answers. This technique works well with deterministic time and space measures and, using the results of Immerman [Imm88] and Szelepcsényi [Sze88], nondeterministic space as well.

For nondeterministic time, we can still do a simulation but can no longer negate the answer directly. Cook [Coo72] uses several layers of a translation argument to reduce the problem to the deterministic time hierarchy, albeit with a weaker separation. Seiferas, Fischer and Meyer [SFM78] extend Cook's work to achieve near optimal bounds.

Now consider a measure like BPTIME where we do have complementation. The Hartmanis-Stearns approach still fails to work because we cannot directly simulate. BPTIME is an example of a semantic non-syntactic measure, 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. A true BPTIME hierarchy still remains a challenging open problem.

In recent years a number of researchers have shown hierarchies for BPTIME and other semantic measures if we allow the classes to have a small amount of advice, i.e., additional information that depends only on the length of the input. A series of papers by Boaz Barak, Lance Fortnow, Rahul Santhanam, Luca Trevisan, Dieter van Melkebeek and Konstantin Perveyshev [Bar02, FS04, FST04, vMP06] lead to the result that every reasonable semantic measure has a hierarchy where both machines have a single bit of advice, for example

$$\mathsf{BPTIME}(n^{3.14})/1 \subsetneq \mathsf{BPTIME}(n^{3.141})/1.$$

This survey will review the work on time hierarchies with a focus on those recent results on the semantic hierarchies, as well as some related work on average-case hierarchies. We hope this survey serves not as a wrap-up of final results in a field but a starting point to find some techniques to remove that pesky last bit of advice.

[1] © Lance Fortnow and Rahul Santhanam, 2006.

[2] Department of Computer Science, University of Chicago `fortnow@cs.uchicago.edu`.
[3] School of Computing Science, Simon Fraser University `rsanthan@cs.sfu.ca`.

2 Notation and Preliminaries

We assume the reader familiar with the basic notions in computational complexity. We define a series of complexity measures on multitape Turing machines.

- $\mathsf{DTIME}(t(n))$ is the set of languages accepted by deterministic machines using time $O(t(n))$.
- $\mathsf{NTIME}(t(n))$ is the set of languages accepted by nondeterministic machines using time $O(t(n))$.
- $\mathsf{RTIME}(t(n))$ is the set of languages accepted by probabilistic machines using time $O(t(n))$ where strings in the language are accepted with probability at least one-half and strings outside the language are always rejected.
- $\mathsf{BPTIME}(t(n))$ is the set of languages accepted by probabilistic machines using time $O(t(n))$ where strings in the language are accepted with probability at least two-thirds and strings outside the language are accepted with probability at most one-third.

For definitions of other complexity measures, refer to the corresponding classes in the Complexity Zoo (`http://complexityzoo.com`).

We distinguish between *semantic* complexity measures and the special case of *syntactic* complexity measures. Syntactic measures are semantic measures for which there is an effective enumeration of machines defining languages in the measure. More formally, we identify each semantic measure with a recursive enumeration $\mathbb{M}$ of machines and a partial "answer" function $A : \mathbb{M} \times \{0,1\}^* \to \{0,1,?\}$. For a machine $M \in \mathbb{M}$ and an input $x \in \{0,1\}^*$, $A(M,x)$ is defined if M halts on x. $A(M,x) = 1$ is to be interpreted as "M accepts x", $A(M,x) = 0$ as "M rejects x" and $A(M,x) =$ "?" as "M doesn't satisfy the promise on x".

Syntactic measures are those for which any machine $M \in \mathbb{M}$ which halts on all inputs has either $A(M,x) = 1$ or $A(M,x) = 0$ for any x. DTIME and NTIME are syntactic measures because each halting DTIME or NTIME machine always outputs an answer on an input - it either accepts or rejects. On the other hand, BPTIME is a semantic non-syntactic measure because there are probabilistic machines which accept with probability $1/2$ and reject with probability $1/2$ - such machines do not give an unambiguous answer on an input.

Sometimes, a non-syntactic class may coincide with a syntactic class. For instance, by results of Lund, Fortnow, Karloff and Nisan [LFKN92], and Shamir [Sha92], $\mathsf{IP} = \mathsf{PSPACE}$, where IP is the class of languages with polynomial-time interactive proofs, defined using the non-syntactic measure IPTIME.

We need to define a notion of advice for semantic measures. $L \in \mathsf{CTIME}(t)/a(n)$ 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 $\mathsf{CTIME}(t)$-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 $\mathsf{CTIME}(t)$-machine on input $< y, x >$ where $y \neq s(|x|)$ which differs from the original definition of advice of Karp and Lipton [KL82] for non-syntactic measures. Proving a hierarchy theorem with advice in the Karp-Lipton model would imply a hierarchy theorem without advice.

A language L is in i.o.-CTIME (t) for a complexity measure CTIME if there is a CTIME machine M which for infinitely many input lengths n, runs in time t and decides L correctly on all inputs of length n.

A language L is in $heur_p$-CTIME (t) for a complexity measure CTIME if there is a CTIME machine M which for each input length n, for at least a fraction p of inputs of length n, runs in time t and decides L correctly on those inputs.

In this survey, the time bounds t we consider are by default constructible, which means that there is a deterministic Turing machine which given input 1^n, outputs $t(n)$ within $t(n)$ steps.

3 Deterministic and Nondeterministic Hierarchies

In this section, we recapitulate the techniques used to prove hierarchies for deterministic and nondeterministic classes.

3.1 Simulation and Diagonalization

The most basic approach to show hierarchies uses simulation and diagonalization. Say we want to show a hierarchy for deterministic time. This means that we need to prove, for time bounds t and T, where T does not grow too much faster than t, that there is a language L in $\mathsf{DTIME}(T) \setminus \mathsf{DTIME}(t)$. Consider an effective enumeration of DTIME machines halting in time t. Assign to each input x a machine $M(x)$ in the enumeration in some easily computable fashion so that each machine in the enumeration has some input assigned to it. We define our language L using a machine M which, given input x, simulates $M(x)$ on x and "does the opposite", i.e., it accepts if $M(x)$ rejects and rejects if $M(x)$ accepts. The machine M can "do the opposite" since DTIME is closed under complement.

L is not in $\mathsf{DTIME}(t)$ - for any DTIME machine M' operating in time t, L differs from $L(M')$ on the (non-empty) set of inputs assigned to M'. Thus we have shown the required lower bound on the complexity of L. The upper bound depends on the complexity of simulation of machines in the enumeration. If $\mathsf{DTIME}(t)$ machines can be simulated in time T, we obtain the required upper bound on the complexity of L.

The above approach was used in the foundational papers [HS65, HS66] to establish hierarchies for time-bounded deterministic computation.

Theorem 1. *[HS65, HS66] For any functions t and T such that T is time-constructible and $t\log(t) = o(T)$, $\mathsf{DTIME}(t) \subsetneq \mathsf{DTIME}(T)$.*

The $t\log(t)$ term in the above result arises from the complexity of simulation of deterministic machines - any k-tape deterministic Turing machine running in time t can be simulated by a 2-tape Turing machine running in time $t\log(t)$ [HS66].

The simulation and diagonalization approach works for any complexity measure which is syntactic and is closed under complement. Thus, it works also for deterministic and nondeterministic space [Imm88, Sze88], and unbounded-error probabilistic time [Gil77]. However, it does not work for nondeterministic time which is not known to be closed under complement.

3.2 Stronger Separations

From Theorem 1, we only get that there is some language $L \in \mathsf{DTIME}(T)$ such that for any $\mathsf{DTIME}(t)$ machine M, there are infinitely many inputs on which M outputs the wrong answer[4].

[4]If there were only finitely many such inputs, we could construct a new machine M' from M into which these inputs are hard-coded, contradicting Theorem 1.

Often, we would like any $\mathsf{DTIME}(t)$ machine M to fail on L in a stronger way, so that the set of inputs on which mistakes are made is "non-sparse" in some sense. This question has been extensively studied; some of the stronger notions of lower bounds that have been considered are:

1. Almost-every-length lower bounds: For any $\mathsf{DTIME}(t)$ machine M and every large enough input length n, there is some input of length n on which M fails to compute L.

2. Lower bounds against advice: For any $\mathsf{DTIME}(t)$ machine M, M fails to compute L even when it is given access to a small advice string.

3. Average-case lower bounds: For any $\mathsf{DTIME}(t)$ machine M, there are infinitely many input lengths n such that mistakes are made on at least $s(n)$ fraction of inputs, for some function $0 \leqslant s(n) \leqslant 1$.

4. Almost-everywhere lower bounds: Any machine M that computes L correctly takes time more than t on all but finitely many inputs.

When the upper bound is of the same kind as the lower bound, we call the corresponding result a hierarchy theorem; when the upper bound is of a more restrictive kind than the lower bound, we call the result a separation theorem. Thus for each of the above notions, a separation theorem implies a hierarchy theorem, but the converse is not necessarily true.

The notions above are all independent, and we can sometimes prove lower bounds in two or more of them simultaneously. A simple application of the simulation-diagonalization paradigm already gives almost-every-length separations against advice with the same parameters as before. We assign "diagonalizable inputs" to machines in a careful way so that for any machine M and any advice string w, for all but finitely many input lengths n, there is an input of length n on which we diagonalize against M with advice string w. Implementing this idea, we obtain the following folklore result:

Theorem 2. *For any functions t and T such that $t\log(t) = o(T)$ and T is time-constructible, there is a language $L \in \mathsf{DTIME}(T)$ such that $L \notin i.o.\mathsf{DTIME}(t)/(n - \log(n))$.*

Obtaining average-case and almost-everywhere separations takes considerably more work. The first average-case separation was obtained by Wilber [Wil83]. Goldreich and Wigderson [GW00] obtained a separation with a much better inapproximability factor. They state their result for exponential-time classes but their technique can be seen to work for any time-constructible bound.

Theorem 3. *There is a universal constant k such that the following holds: Let T be any time-constructible function, and $R = min(T, 2^{n/2})$. Then there is a language L in $\mathsf{DTIME}(T^k)$ such that $L \notin i.o.heur_{1/2+1/R} - \mathsf{DTIME}(T)/\log(R)$.*

One shortcoming of Theorem 3 is that the gap between the lower bound and the upper bound is larger than we might hope. This gap can be reduced at the cost of a worse inapproximability factor.

An almost-everywhere separation for deterministic time was obtained by Geske, Huynh and Seiferas [GHS91].

Theorem 4. *[GHS91] For any constructible functions t and T such that $t\log(t) = o(T)$, there is a language L in $\mathsf{DTIME}(T)$ such that any deterministic machine accepting L takes time more than t on all but finitely many inputs.*

Results analogous to Theorem 2, Theorem 3 and Theorem 4 hold for any syntactic measure closed under complement.

3.3 Indirect Diagonalization

The simulation-diagonalization approach doesn't apply directly to the case of nondeterministic time. The problem is that nondeterministic time is not known to be closed under complement, hence it is unclear how to define a nondeterministic machine that "does the opposite".

The problem can be circumvented through an "indirect diagonalization". We assume that a hierarchy theorem does not hold and then find a way to "boost" this assumption until we derive a contradiction to some other hierarchy proved using a direct diagonalization. Boosting can be done using a translation method [RF65, Iba74]. Given a simulation of one resource-bounded class in another, translation gives a corresponding simulation where the two classes use proportionately more resources.

Lemma 5. *If* $\mathsf{BTIME}(t_1) \subseteq \mathsf{CTIME}(t_2)$ *for some complexity measures* BTIME *and* CTIME *and time bounds* t_1 *and* t_2*, then for any time-constructible function* $f(n) \geqslant n$, $\mathsf{BTIME}(t_1(f)) \subseteq \mathsf{CTIME}(t_2(f))$.

The proof of Lemma 5 is simple. Suppose we want to show that a language $L \in \mathsf{BTIME}(t_1(f))$ can be solved in $\mathsf{CTIME}(t_2(f))$. Consider a padded version L' of L, where an input $x' \in L'$ if and only if x' consists of a string $x \in L$ followed by $f(|x|) - |x|$ consecutive 1's. Using time-constructibility of f, $L' \in \mathsf{BTIME}(t_1)$, since a machine for L' can be defined which simulates a machine for L on the portion of the input with the padding removed. By assumption, $L' \in \mathsf{CTIME}(t_2)$. Now a $\mathsf{CTIME}(t_2(f))$ machine for L can be defined which pads the input x to L with $f(|x|) - |x|$ 1's (again using time-constructibility of f) and then runs the $\mathsf{CTIME}(t_2)$ machine for L' on the padded input.

Without using translation at all, we can already prove that $\mathsf{NTIME}(n) \subsetneq \mathsf{NTIME}(n^n)$, as follows. If $\mathsf{NTIME}(n^n) \subseteq \mathsf{NTIME}(n)$, then $\mathsf{DTIME}(n^n) \subseteq \mathsf{NTIME}(n^n) \subseteq \mathsf{NTIME}(n) \subseteq \mathsf{DTIME}(2^{O(n)})$, where we use the facts that determinism is a special case of nondeterminism, and that a nondeterministic computation can be simulated deterministically with an exponential slowdown. Thus we have $\mathsf{DTIME}(n^n) \subseteq \mathsf{DTIME}(2^{O(n)})$, which is a contradiction to Theorem 1.

But the nondeterministic time bounds we separate in this manner are very far apart; the translation method allows us to reduce this gap. Suppose T is a constructible function such that $T^{(k)}(n) = \omega(n^n)$ for some constant k. If $\mathsf{NTIME}(T) \subseteq \mathsf{NTIME}(n)$, then by k applications of Lemma 5, we have that $\mathsf{NTIME}(T^{(k)}) \subseteq \mathsf{NTIME}(n)$, which is a contradiction to the separation we proved in the preceding paragraph. Thus we get a separation of $\mathsf{NTIME}(T)$ and $\mathsf{NTIME}(n)$ even for certain time bounds T which are sub-exponential (just by choosing k large enough). However, the time bounds T for which we derive separations in this way are not even close to polynomial in t, let alone $O(t\mathsf{polylog}(t))$ as in Theorem 1. A polynomially bounded function composed with itself a constant number of times remains polynomially bounded, rather than reaching the exponential heights we covet.

Steve Cook [Coo72] was the first to derive a hierarchy theorem for nondeterministic time where T was polynomial in t. He used the basic framework sketched above. The key additional idea was that he could apply Lemma 5 a *non-constant* number of times by using the fact that NTIME is a syntactic measure. We do not go into the details of his proof, since later methods achieved better parameters, and restrict ourselves to stating his result:

Theorem 6. *For any reals a and b such that $1 \leqslant a < b$,* $\mathsf{NTIME}(n^a) \subsetneq \mathsf{NTIME}(n^b)$.

Seiferas, Fischer and Meyer [SFM78] and Žák [Ž83] gave a nondeterministic hierarchy theorem with better parameters by using a different indirect diagonalization argument.

Theorem 7. *For any time-constructible functions t and T such that $t(n+1) = o(T(n))$,* $\mathsf{NTIME}(t) \subsetneq \mathsf{NTIME}(T)$.

To compare with Theorem 6, note that when t is a polynomial, Theorem 7 implies a separation of $\mathsf{NTIME}(t)$ and $\mathsf{NTIME}(T)$ for ant T such that $t = o(T)$. Thus not only does Theorem 7 beat Theorem 6, but it is also gives a tighter separation than Theorem 1 does for deterministic time in the polynomial time range! This tightness owes to the fact that 2-tape nondeterministic Turing machines can simulate k-tape machines with only a constant factor slowdown [BGW70].

We now give a sketch of the proof of Theorem 7, choosing Žák's proof for its relative simplicity.

Suppose we want to define a nondeterministic machine M that diagonalizes against a machine M_i running in time t on input x. Since M is nondeterministic, it cannot directly "do the opposite", and since we'd like M to run in time only slightly more than M_i, it cannot simulate M_i deterministically and then "do the opposite". However, for the purpose of deriving a hierarchy, M does not need to differ from M_i on the specific input x, rather it suffices that there is some input on which the machines differ. For any $n <= j < m = n^{t(n)}$, M on input 1^j simulates M_i on input 1^{j+1}, accepting if M_i accepts and rejecting if M_i rejects. The existence of an effective enumeration for nondeterministic machines allows us to define a machine performing such a simulation. When $j = m$, M on input 1^j simulates M_i on input x *deterministically*, accepting if M_i rejects and rejecting if M_i accepts. Note that since $m = n^{t(n)}$, M has enough time to perform the trivial exponential-time deterministic simulation of M_i on input 1^n. Essentially, what we are doing is deferring the diagonalization step by linking M and M_i together on larger and larger inputs until M has an input large enough that it can actually "do the opposite" deterministically.

To see that this works, assume that $M(1^j) = M_i(1^j)$ for all $n \leqslant j \leqslant m$. By definition of M, we also have that $M(1^j) = M_i(1^{j+1})$ for all $n \leqslant j < m$. Thus we have that $M_i(1^n) = M(1^n) = M_i(1^{n+1}) = \ldots = M(1^{m-1}) = M_i(1^m) = M(1^m)$, but from the definition of M, $M(1^m) \neq M_i(1^n)$, thus we have a contradiction.

There are a few details that remain to be worked out. We have shown how to diagonalize against a single machine M_i but in fact we have to diagonalize against all such machines. Also, we haven't specified how to choose the sequence of inputs. Both problems can be solved by diagonalizing against M_i on sequences of inputs starting with input of the form $x_0 = 1^i 0 1^r 0$ for any $r > 0$, with the jth input in the sequence being $x_j = 1^i 0 1^r 0 1^j$. M behaves arbitrarily on inputs not of this form[5]. As far as the time taken by M is concerned, if we are diagonalizing against time t machines, then M can be made to run in time T for any constructible T such that $t(n+1) = o(T(n))$. This is because M needs to simulate a time t nondeterministic machine on an input of length one larger than its own input length, for inputs x_j in a sequence where $j < m$. When $j = m$, M needs to perform the deterministic simulation of M_i on an exponentially smaller input, and this can be done in linear time.

The only property of nondeterministic time we used in the proof of Theorem 7 is that it has an effective enumeration. Thus Theorem 7 also works for other syntactic complexity measures not known to be closed under complement, such as $\Sigma_k - \mathsf{TIME}$ for any positive k.

[5]With a more careful coding, the diagonalizing language $L(M)$ can in fact be made unary.

Theorem 7 does not have very good parameters when t is large. For instance, it does not show that $\mathsf{NTIME}(2^{2^{n+1}}/\log(n)) \not\subseteq \mathsf{NTIME}(2^{2^n})$, though the corresponding result for deterministic time follows easily from Theorem 1. Rackoff and Seiferas [RS81] showed that this deficiency holds for any relativizing method, by giving an oracle with respect to which $\mathsf{NTIME}(2^{2^{n+1}}/\log(n)) = \mathsf{NTIME}(2^{2^n})$. Every hierarchy theorem proved to date has a relativizing proof, so a novel technique would be required to get better parameters.

Another weakness of the technique of Theorem 7 is that it does not yield most of the stronger kinds of lower bounds we considered in Subsection 3.2. The best almost-everywhere, almost-every-length and average-case separations known for nondeterministic time (when the upper bound is worst-case) are weak ones that can be proved by using padding to boost a presumed inclusion a constant number of times and then using direct diagonalization to derive a contradiction. Allender, Beigel, Hertrampf and Homer [ABHH93] showed that for the case of almost-everywhere separations, this is the best result that can be obtained using relativizing methods. For separations against advice, the situation is a little better. Fortnow, Santhanam and Trevisan [FST05] give a separation against a small amount of advice by using a variant of Žák's technique.

Theorem 8. *For any real number $a \geqslant 1$, there exists a real $b > a$ such that* $\mathsf{NTIME}(n^b) \not\subseteq \mathsf{NTIME}(n^a)/\log(n)^{1/2a}$.

4 Hierarchies for Randomized Classes

4.1 Hierarchies via Optimal Algorithms

BPTIME and RTIME are natural complexity measures for which one would like to prove hierarchies. Such results would have particular significance because efficient solvability in probabilistic time is often identified with feasibility, and hence the existence of hierarchies implies that there are degrees of feasibility when solving computational problems.

The indirect diagonalization technique used to prove Theorem 7 doesn't work for these measures. The technique makes essential use of the existence of an efficient measure, while BPTIME and RTIME are non-syntactic.

The method of boosting a presumed collapse using translation to yield a contradiction to direct diagonalization, outlined at the beginning of Section 3.3, does work, since the assumption of an efficient enumeration is not used there. This method gives the best hierarchy known thus far for completely uniform probabilistic classes [KV87].

Theorem 9. *Let t and T be time-constructible functions such that there exists a constant k for which $T^k(t) = 2^{\omega(t)}$. Then* $\mathsf{BPTIME}(t) \subsetneq \mathsf{BPTIME}(T)$.

The absence of an efficient enumeration is a fundamental obstacle to proving tight hierarchies, since known diagonalization results with good parameters such as Theorem 1 and Theorem 7 use simulation in an essential way. In a breakthrough, Barak [Bar02] showed that this obstacle can be overcome to show hierarchies with advice (as opposed to hierarchies for completely uniform classes) for probabilistic time with two-sided error.

Theorem 10. *For any reals $1 \leqslant a$, there is a real b such that* $\mathsf{BPTIME}(n^b)/\log(\log(n)) \not\subseteq \mathsf{BPTIME}(n^a)/\log(n)$.

The same result holds with the complexity measure BQTIME corresponding to time complexity of quantum Turing machines.

Barak introduced a new paradigm for proving hierarchy theorems, based on *optimal algorithms*. An optimal algorithm for a language L is an algorithm that is at worst polynomially slower than *any* algorithm for L. Let k be a universal constant such that if there is an algorithm for L running in time t, then the optimal algorithm runs in time less than t^k. If T is the time taken by the optimal algorithm, then we get that L is solvable in time T but not in time $T^{1/k}$.

Modulo the existence of an optimal algorithm, this already gives us a hierarchy of a sort, albeit for a mysterious time function T. To get hierarchies for polynomial time bounds, we could hope to use translation to "scale" the separation downward, as long as T is super-polynomial. The obstacle here is that there is no reason to believe that T is constructible, and Lemma 5 requires constructibility of T. Barak gets around this obstacle by using a small amount of advice to encode a good approximation to T - this is why he only gets a hierarchy with advice and not a fully uniform hierarchy. He shows how to encode any T that is at most exponential in n with at most $\log(\log(n))$ bits of advice. Since the upper bound uses advice, we must take care that the lower bound is against classes with at least as much advice, since otherwise a separation can be shown by a simple counting argument.

Barak's main technical contribution was to construct probabilistic optimal algorithms with two-sided error for non-trivial languages[6]. Specifically, he showed how to construct an optimal algorithm for any EXP-complete language L. The construction relies on the existence of *instance checkers* for EXP-complete languages, which follows from the famous results on probabilistically checkable proofs [BFL91, AS98]. Informally, an instance checker is a procedure that given an input x and oracle access to a machine M, distinguishes with high confidence between the case that M decides L correctly on all inputs and the case that M decides L incorrectly on input x. The optimal algorithm, given input x, cycles among all probabilistic machines of small size, searching for one that decides L correctly. It uses the instance checker to weed out those machines that decide L incorrectly on x. Of course, the optimal algorithm doesn't know a priori how long it should run each machine, so it just tries different running times $1, 2, 4, 8 \ldots$ in succession. If there is some machine M_i running in time T deciding L, the optimal algorithm will only spend time polynomial in T before trying M_i for $2^{\lceil \log(T) \rceil}$ steps, at which point it will halt with high probability with the right answer. Also, by the property of the instance checker, there is only a very small probability that the optimal algorithm outputs the wrong answer before this stage. These two facts together establish the correctness of the optimal algorithm.

To complete the proof of a hierarchy, there are two possibilities to consider. If the optimal algorithm runs in polynomial time, then $L \in$ BPP and hence EXP = BPP, using the fact that L is EXP-complete. In this case, a hierarchy with advice follows from the corresponding hierarchy for deterministic exponential time. In the other case, the time function T of the optimal algorithm is super-polynomial and we can use translation as described earlier to obtain a hierarchy with small advice.

To make progress towards a uniform hierarchy, we would like to reduce the amount of advice in the upper bound as much as possible. If the upper bound used no advice at all, then we would have a separation against advice, and in particular a hierarchy. Fortnow and Santhanam [FS04] showed how to reduce the advice to 1 bit. Their proof followed the same basic framework as Barak's, with the difference being in the translation step. They showed how to accomplish the translation

[6] Any language in BPP trivially has such an optimal algorithm.

using just 1 bit of advice to indicate whether the amount of padding is large enough that the optimal algorithm runs in time polynomial in the length of the padded input. Goldreich, Sudan and Trevisan [GST04] abstracted out their idea to show that any separation of a time-bounded class with less than logarithmic advice against a class using slightly more advice can be translated to a separation of the first class (using slightly less time) with 1 bit of advice against the second class (using slightly less time) with slightly more advice. Thus we obtain the following separation result which improves on Theorem 10.

Theorem 11. *For any real $a \geqslant 1$, there is a real b such that* $\mathsf{BPTIME}(n^b)/1 \not\subseteq \mathsf{BPTIME}(n^a)/\log(n)$.

We get a tighter hierarchy using translation if we just want a hierarchy with 1 bit of advice and not a separation against $\log(n)$ bits of advice.

Corollary 12. *For any reals $1 \leqslant a < b$,* $\mathsf{BPTIME}(n^a)/1 \subsetneq \mathsf{BPTIME}(n^b)/1$.

Barak's paradigm does have the disadvantage that it does not give good parameters if we are interested in super-polynomial time bounds. We discuss later how other more generic methods do not suffer from this flaw.

4.2 An Average-Case Hierarchy

In the proof of Theorem 10, the need for advice in the upper bound arises because we do not know how to efficiently compute the time T taken by the optimal algorithm. Perhaps by choosing a language L with more structure, an optimal algorithm for L can be defined for which the time function can be computed efficiently? Fortnow and Santhanam [FS04] explored this path, and though they did not succeed in getting a fully uniform hierarchy, they did prove an average-case hierarchy for BPTIME in the range of polynomial time bounds.

The basic obstacle towards computing the time function is that it represents the *maximum* of the time taken over all possible inputs of a certain length, and it is unclear how to compute such a maximum efficiently. We have no a priori reason to believe that there isn't a wide variation between running times on different inputs, and it seems hard to tell which input is the "worst-case" one. The main idea of Fortnow and Santhanam is that for certain languages, the maximum can be estimated well by the *average* running time. They observe that such an estimation works for languages that are *random self-reducible*. Random self-reducible languages are those for which the answer on any given input x can be computed efficiently from the answers on inputs $f_1(x), f_2(x) \ldots f_k(x)$ generated randomly form x, where each $f_i(x), i = 1 \ldots k$ is distributed uniformly over inputs of length $|x|$. Intuitively, this property guarantees that the time for the worst-case input is not worse by more than a polynomial factor than the time for a random input.

It is known that there are EXP-complete and PSPACE-complete languages that are randomly self-reducible. Fortnow and Santhanam choose a specific PSPACE-complete language L which is random self-reducible, and define an optimal algorithm for L which incorporates the random self-reduction on top of the instance checker. They then show how to estimate the time function of this optimal algorithm to within a polynomial factor by running the optimal algorithm on randomly chosen inputs and then using the random self-reducibility property to argue that the time taken on these inputs provides a good estimate of the worst-case time. With such an estimate in hand,a modified translation lemma can be used to prove the average-case hierarchy. An average-case hierarchy is obtained rather than a worst-case one because only an *estimate* of the time function of

the optimal algorithm is computed; if the time function could be computed exactly, then we would indeed have a fully uniform hierarchy.

Theorem 13. *For any real $1 \leqslant a$, there is a real $b > a$ and constants $c > d$ such that there is a language L for which $L \in heur_{1-1/n^c} - \mathsf{BPTIME}(n^b)$ but $L \notin heur_{1-1/n^d} - \mathsf{BPTIME}(n^a)$.*

4.3 Hierarchy for RTIME with 1 Bit of Advice

Barak's proof only works for complexity measures that are closed under probabilistic Turing reductions with two-sided error, such as BPTIME or BQTIME. However, the general paradigm of hierarchies via optimal algorithms is potentially applicable to other classes. Fortnow, Santhanam and Trevisan [FST05] used this paradigm to give hierarchies with advice for probabilistic time with one-sided error.

Theorem 14. *For any real $1 \leqslant a$, there is a real $b > a$ such that $\mathsf{RTIME}(n^b)/1 \not\subseteq \mathsf{RTIME}(n^a)/\log(n)^{1/2a}$.*

As in Corollary 12, a tight hierarchy with one bit of advice can be derived from this separation result using a translation lemma.

The proof of Theorem 14 proceeds by defining a probabilistic optimal algorithm with one-sided error for the NP-complete problem SAT. Given the existence of such an algorithm, there are two cases to consider. If the optimal algorithm for SAT runs in polynomial time, then since SAT is NP-complete, we have that NP = RP. In this case, using Theorem 8, we get a hierarchy with 1 bit of advice. If the optimal algorithm takes super-polynomial time then an advice-efficient translation lemma [FS04, GST04] can be used to derive a hierarchy with 1 bit of advice for the polynomial time range.

It remains to define the optimal algorithm. The optimal algorithm is a one-sided probabilistic version of Levin's deterministic optimal algorithm for SAT [Lev73]. Given a formula ϕ as input, we cycle over probabilistic machines of small size, searching for one that decides ϕ correctly. Again, we don't know a priori how long to run a machine, hence we try running times $1, 2, 4 \ldots$. If a machine M_i run for t steps accepts on ϕ, we use M_i restricted to t steps to search for a satisfying assignment through downward self-reducibility. Thus the optimal algorithm only accepts on ϕ when it can find a proof that ϕ is satisfiable. If there is a probabilistic machine M_i for SAT with exponentially small error running in T steps, then the optimal algorithm only uses time polynomial in T before running M_i for $2^{\lceil \log(T) \rceil}$ steps and searching for a satisfying assignment if M_i accepts. In this case, the optimal algorithm accepts with high probability within $poly(T)$ steps. On the other hand, if ϕ is not satisfiable, the optimal algorithm never accepts.

5 Hierarchies for General Semantic Classes

5.1 Hierarchies with Small Advice

There are several interesting complexity measures, e.g., NTIME∩coNTIME and ZPTIME, for which the results in Section 3 and Section 4 do not establish hierarchies. The techniques of Section 3 do not apply to these measures because they are not syntactic. On the other hand, there don't seem to be natural examples of languages which have optimal algorithms according to these measures, so as to enable us to apply the techniques of Section 4. The paradigm of hierarchies via

optimal algorithms has the disadvantage that it applies to very specific kinds of measures. Just as the simulation-diagonalization paradigm and the indirect diagonalization paradigm apply to broad classes of measures, i.e., syntactic measures closed under complement and general syntactic measures respectively, it would be convenient to have a general technique for establishing hierarchies for general semantic measures.

Fortnow, Santhanam and Trevisan [FST05] addressed this question and showed hierarchies with small advice for a general class of "reasonable" semantic measures. Here, a reasonable measure CTIME is any measure such that $\mathsf{DTIME}(t) \subseteq \mathsf{CTIME}(t) \subseteq \mathsf{DTIME}(2^{O(t)})$ for constructible t, and CTIME is closed under deterministic transductions. In particular, commonly encountered semantic measures such as $\mathsf{BPTIME}, \mathsf{RTIME}, \mathsf{ZPTIME}, \mathsf{BQTIME}, \mathsf{NTIME} \cap \mathsf{coNTIME}, \mathsf{UTIME}$, etc. satisfy these properties.

Theorem 15. *Let* CTIME *be any reasonable semantic measure. For any real* $a \geqslant 1$*, there is a real* $b > a$ *and a function* $h(n) = O(\log(n) \log\log(n))$ *such that* $\mathsf{CTIME}(n^a)/h(n) \subsetneq \mathsf{CTIME}(n^b)/h(n)$.

By using the generic advice reduction technique of Goldreich, Sudan and Trevisan [GST04], a hierarchy with 1 bit of advice can be derived for quasi-polynomial time.

Theorem 16. *Let* CTIME *be any reasonable semantic measure. For any reals* $1 \leqslant a < b$*,* $\mathsf{CTIME}(2^{\log(n)^a}) \subsetneq \mathsf{CTIME}(2^{\log(n)^b})$.

We provide here only a brief sketch of the proof of Theorem 15. The main idea is to use a non-uniform version of Barak's paradigm. To derive a hierarchy theorem with advice, it suffices to define a non-uniform optimal algorithm for a non-trivial language, i.e., different algorithm can be used for different input lengths. Such algorithms exist for any reasonable semantic measure. Given a reasonable semantic measure CTIME, we take a language L that is known to require deterministic exponential time (existence of such a language is shown by direct diagonalization), and try to speed up CTIME decidability of L quadratically by using a small additional amount of advice. If we don't succeed, then there is a time bound t such that L is decidable in $\mathsf{CTIME}(t)$ but not in $\mathsf{CTIME}(\sqrt{t})$ on inputs of length n. In this case, an advice-efficient translation lemma can be used to get a hierarchy with advice. If we do succeed in speeding up the decidability, then we try to speed it up 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. Thus, at some point, we will find ourself in the first case and can apply the translation lemma. Much of the work in the proof goes towards reducing the advice required.

The proof of Theorem 15 has the positive feature that it can be used to show hierarchies for non-polynomial time bounds too, unlike the proof of Theorem 10.

Theorem 17. *Let* CTIME *be any reasonable semantic measure. For any constructible time bound* t *such that* $n \leqslant t \leqslant 2^n$*, there is a constant* $\epsilon > 0$ *and an advice bound* $h(n) = O(\log(t) \log\log(t))$ *such that* $\mathsf{CTIME}(t^\epsilon)/h(n) \subsetneq \mathsf{CTIME}(t)/h(n)$.

The same techniques work for syntactic measures such as DTIME and NTIME, since a syntactic measure is just a special case of a semantic measure. However, as shown in Section 3, stronger results can be proved for these measures using direct or indirect diagonalization.

5.2 Hierarchies with 1 Bit of Advice

Van Melkebeek and Pervyshev [vMP06] simultaneously improved Theorems 15 and 16 to show that any reasonable semantic measure has a hierarchy in the polynomial-time range with 1 bit of advice.

Theorem 18. *Let* CTIME *be any reasonable semantic measure. For any reals* $1 \leqslant a < b$, $\mathsf{CTIME}(n^a)/1 \subsetneq \mathsf{CTIME}(n^b)/1$.

The technique used to prove Theorems 15 and 16 seems to inherently require $> \log(n)$ bits of advice, and the advice reduction lemma [GST04] only allows us to reduce $\log(n)$ bits to 1 bit of advice. Hence a new technique is required. Van Melkebeek and Pervyshev observe that the technique of Fortnow, Santhanam and Trevisan can be viewed as an adaptation of Cook's proof of Theorem 6 to the context of semantic measures. They then show that an analogous adaptation of Žák's proof of Theorem 7 to the context of semantic measures yields Theorem 18.

We summarize the ideas of their proof. We use the RTIME measure in our summary since this is a natural example of a measure which is neither syntactic nor closed under complement, as in the general case. Suppose we want to diagonalize against a probabilistic machine M_i which uses 1 bit of advice on inputs of length n. Two issues arise when we try to do a direct diagonalization. First, there is the problem that RTIME is not closed under complement, so we can't "do the opposite". Second, even if we were dealing with a complexity measure closed under complement, our diagonalizing machine should always satisfy the promise when given the correct advice, even if there is an advice bit on which M_i doesn't satisfy the promise. Of course, if M_i doesn't satisfy the promise on an advice bit b, we don't need to diagonalize against M_i with advice bit b, however we don't know a priori on which advice bit(s) M_i satisfies the promise for all inputs of length n and on which it doesn't. To represent all the possibilities, 2 bits of advice are required, which is more than we can afford in our upper bound. In general, if we are diagonalizing against c bits of advice for M_i, we require 2^c bits to represent whether M_i satisfies the promise on different advice strings or not, which is exponentially more than we can afford.

Van Melkebeek and Pervyshev tackle both issues by encoding information in the input length. Given a starting input length n and an advice bit b, they define a polynomially smaller input length n_1 such that n and b can be uniquely recovered from n_1 (such a recovery isn't possible for all n, but we can choose to deal only with n of a special form). On an input x of length n_1, the diagonalizing machine M simulates M_i on input $x1^{n-n_1}$ with advice b, accepting if M_i accepts and rejecting if M_i rejects. However, M only does the simulation if its advice bit is set to 1, indicating that M_i satisfies the promise on all inputs of length n. If the advice bit is set to 0, M just rejects all inputs of length n_1. We think of M as "copying" the behavior of M_i on length n to length $m(n, b)$. By "spreading" this copying using a different input length for each advice bit of M_i, we avoid the need for M to have a larger advice string than M_i.

M does the copying repeatedly. It copies the behavior of M_i on length n_1 to a polynomially smaller length n_2 from which n_1 and the advice bit b_1 for M_i on inputs of length n_1 can be uniquely recovered. In general, a sequence $n_0 = n, n_1, n_2 \ldots n_k$ is created such that the behavior of M_i on input length n_i is copied on to input length n_{i+1}, for $i = 0 \ldots k$. The copying bottoms out at a length n_k for which n is exponentially larger than n_k. At this point, diagonalization replaces copying, just as in Žák's argument. The proof that this works is analogous to Žák's; the main additional work is to show that the copying can be done repeatedly while still preserving the recoverability of n_i and b_i from n_{i+1}.

Van Melkebeek and Pervyshev actually show how to diagonalize against a bits of advice using just 1 bit of advice, for any constant a. However, though Theorem 18 subsumes previous work on hierarchies with one bit of advice for specific semantic measures, Theorem 10 and Theorem 14 still achieve the best known separations for BPTIME and RTIME respectively. Van Melkebeek and Pervyshev give alternate proofs of these results, using a hybrid of their ideas and previous

techniques.

Grigoriev, Hirsch and Pervyshev [GHP05] use related techniques to derive hierarchies for cryptographic function inversion with 1 bit of advice.

6 Conclusions and Future Work

We have reviewed how a series of results lead to a near tight hierarchy for essentially every reasonable time measure. The big open question remains to show a separation without advice, for example that $\mathsf{BPTIME}(n^4)$ is not contained in $\mathsf{BPTIME}(n^2)$ but the current methods do not immediately lead to such results. We don't have a separation for any measure not known to be polynomially related to a syntactic measure.

For a given resource bound CTIME and $a < b$ what is the largest amount of advice $f(n)$ where we can prove a separation

$$\mathsf{CTIME}(n^b)/1 \subsetneq \mathsf{CTIME}(n^a)/f(n)?$$

We don't know how to provably increase $f(n)$ beyond constant for general semantic classes. This question remains interesting even for syntactic measures like NTIME and DTIME. Showing that P is not contained in $\mathsf{DTIME}(n^k)/n^k$ will separate BPP from EXP. For NTIME we don't know how to increase $f(n)$ beyond $O(\log n)$.

BPTIME and BQTIME are the only non-syntactic measures for which an average-case hierarchy is known. Do average-case hierarchies hold for general semantic measures?

References

[ABHH93] Eric Allender, Richard Beigel, Ulrich Hertrampf, and Steven Homer. Almost-everywhere complexity hierarchies for nondeterministic time. *Theoretical Computer Science*, 115(2):225–241, 19 July 1993.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM*, 45(1):70–122, 1998.

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

[BC93] Daniel Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice-Hall, 1993.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BGW70] Ronald V. Book, Sheila A. Greibach, and Ben Wegbreit. Time- and tape-bounded Turing acceptors and AFLs. *Journal of Computer and System Sciences*, 4(6):606–621, December 1970.

[Bor94] Bernd Borchert. *Predicate Classes, Promise Classes, and the Acceptance Power of Regular Languages*. PhD thesis, Mathematisches Institut, Universität Heidelberg, Heidelberg, Germany, 1994.

[Can15] Georg Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers*. The Open Court, Chicago, 1915. Translated by P. E. B. Jourdain; a Dover edition (1952) is available.

[Coo72] Stephen 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*, pages 316–324, 2004.

[FST04] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. Promise hierarchies. *Electronic Colloquium on Computational Complexity(ECCC)*, 11(98), 2004.

[FST05] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. Hierarchies for semantic classes. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, 2005.

[GHP05] Dima Grigoriev, Edward Hirsch, and Konstantin Pervyshev. Time hierarchies for cryptographic function inversion with advice. *Electronic Colloquium on Computational Complexity*, 12(76), 2005.

[GHS91] John Geske, Dung Huynh, and Joel Seiferas. A note on almost-everywhere-complex sets and separating deterministic-time-complexity classes. *Information and Computation*, 92(1):97–104, 1991.

[Gil77] John Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GST04] Oded Goldreich, Madhu Sudan, and Luca Trevisan. From logarithmic advice to single-bit advice. *Electronic Colloquium on Computational Complexity, TR04-093*, 2004.

[Gur83] Yuri Gurevich. Algebras of feasible functions. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, November 1983.

[GW00] Oded Goldreich and Avi Wigderson. On pseudorandomness with respect to deterministic observers. In *Electronic Colloquium on Computational Complexity, technical reports*, 2000.

[HH88] Juris Hartmanis and Lane A. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58(1–3):129–142, 1988.

[HJ95] Lane Hemaspaandra and Sudhir Jha. Defying upward and downward separation. *Information and Computation*, 121(1):1–13, 1995.

[HJV93] Lane Hemaspaandra, Sanjay Jain, and Nikolai Vereshchagin. Banishing robust Turing completeness. *International Journal of Foundations of Computer Science*, 4(3):245–265, 1993.

[HR92] Lane Hemachandra and Roy Rubinstein. Separating complexity classes with tally oracles. *Theoretical Computer Science*, 92(2):309–318, 1992.

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

[HS66] Frederick Hennie and Richard Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13(4):533–546, October 1966.

[Iba74] Oscar Ibarra. A hierarchy theorem for polynomial space recognition. *SIAM Journal on Computing*, 3(3):184–187, 1974.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

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

[KV87] Marek Karpinski and Rutger Verbeek. On the monte carlo space constructible functions and separation results for probabilistic complexity classes. *Information and Computation*, 75, 1987.

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

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, 1992.

[RF65] S. S. Ruby and P. C. Fischer. Translational methods and computational complexity. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 173–178. IEEE, 1965.

[Rot95] Jörg Rothe. *On Some Promise Classes in Structural Complexity Theory*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität, Jena, Germany, 1995.

[RRW94] Rajesh Rao, Jörg Rothe, and Osamu Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, 1994. Corrigendum appears in the same journal, 74(1–2):89.

[RS81] Charles Rackoff and Joel Seiferas. Limitations on separating nondeterministic complexity classes. *SIAM Journal on Computing*, 10(4):742–745, 1981.

[SFM78] Joel Seiferas, Michael Fischer, and Albert Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, January 1978.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, 1992.

[Sip82] Michael Sipser. On relativization and the existence of complete sets. In *Proceedings of the 9th International Colloquium on Automata, Languages, and Programming*, pages 523–531. Springer-Verlag *Lecture Notes in Computer Science #140*, July 1982.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, November 1988.

[Tur36] A. Turing. On computable numbers, with an application to the Etscheidungs problem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[vMP06] Dieter van Melkebeek and Konstantin Pervyshev. A generic time hierarchy for semantic models with one bit of advice. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, page To appear, 2006.

[Wil83] Robert Wilber. Randomness and the density of hard problems. In *24th Annual Symposium on Foundations of Computer Science*, pages 335–342, 1983.

[ŽŚ3] Stanislav Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, October 1983.