Hierarchy Theorems for Probabilistic Polynomial Time

Lance Fortnow
Department of Computer Science
University of Chicago
E-mail: fortnow@cs.uchicago.edu

Rahul Santhanam
Department of Computer Science
University of Chicago
E-mail: rahul@cs.uchicago.edu

Abstract

We show a hierarchy for probabilistic time with one bit of advice, specifically we show that for all real numbers $1 \leqslant \alpha < \beta$, $BPTIME(n^\alpha)/1 \subsetneq BPTIME(n^\beta)/1$. This result builds on and improves an earlier hierarchy of Barak using $O(\log\log n)$ bits of advice.

We also show that for any constant $d > 0$, there is a language L computable on average in BPP but not on average in $\mathsf{BPTIME}(n^d)$.

We build on Barak's techniques by using a different translation argument and by a careful application of the fact that there is a PSPACE-complete problem L such that worst-case probabilistic algorithms for L take only slightly more time than average-case algorithms.

1 Introduction

Can we solve more problems given more time? This fundamental question has challenged many complexity theorists since the field started. Hennie and Stearns [HS66] improving on Hartmanis, Lewis and Stearns [SHL65] show a tight hierarchy for time on deterministic multitape machines: For any reasonable time functions T_1 and T_2 with $T_2\log(T_2) = o(T_1)$ there is a language computable in deterministic time $T_1(n)$ and not computable in time $T_2(n)$. We also have similar results for nondeterministic time (see [Coo72, SFM78, Ž́83]).

In this paper we study the question of a time hierarchy for bounded-error probabilistic machines. The results for deterministic and nondeterministic computation have at their core a diagonalization against all smaller time machines. We cannot directly do such a diagonalization for probabilistic classes, since that will break the bounded-error promise when we try to simulate a machine for which the promise does not hold.

Recently Boaz Barak [Bar02] found a different attack on the probabilistic time hierarchy. He looks at the best probabilistic algorithms to compute an EXP-complete language and using a translation argument can build a hierarchy based on these algorithms. However he needs some nonuniform advice to describe an approximation of the running time of that best algorithm. For each constant $d \geqslant 1$, Barak shows that there is a language in $\mathsf{BPP}/\log(\log(n))$

but not in $\mathsf{BPTIME}(n^d)/\log(n)$, and from this he derives a hierarchy theorem for probabilistic polynomial-time machines using $a(n)$ bits of advice, where $\log(\log(n)) \leqslant a(n) \leqslant \log(n)$, using a standard argument.

To prove our first main result, we modify Barak's translation argument. By requiring the pad to conform to a certain format, we can bring the amount of advice required down to one bit, which essentially tells us whether the pad is a good approximation of the time taken by the optimal algorithm.

Theorem 1. *For each constant $d \geqslant 1$, $\mathsf{BPTIME}(n^d)/1 \subsetneq \mathsf{BPP}/1$.*

Using standard translation arguments, from Theorem 1 it follows that for each $1 \leqslant \alpha < \beta$, $\mathsf{BPTIME}(n^\alpha)/1 \subsetneq \mathsf{BPTIME}(n^\beta)/1$.

We attempt to eliminate the advice using an additional idea: for a certain PSPACE-complete language L, we can modify an optimal probabilistic algorithm so that the average-case time of the algorithm is smaller than the worst-case time by at most a polynomial factor. This allows us to estimate the running time, and we can compare this estimate directly with the pad, rather than needing to refer to the advice.

There are two obstacles we run into when we work out his approach. The first obstacle is that we do not know how to use the translation argument when the best algorithm for L runs in polynomial time. However, in this case we can use diagonalization to directly obtain a hierarchy for average-case probabilistic polynomial time.

The second obstacle is that since the estimate of the running time we obtain isn't completely precise, the resulting simulation may not satisfy the bounded-error promise required of a BPP machine. We get around this by randomly perturbing the estimate depending on the input, which ensures that the promise is satisfied for most inputs. As a consequence, we again get a hierarchy for average-case probabilistic polynomial time in the case when the best algorithm for L does not run in polynomial time. Thus we obtain unconditionally a hierarchy theorem which we formalize as:

Theorem 2. *For each constant $d \geqslant 1$, there is a constant $a > 0$ such that $heur_{1-n^{-a}} - \mathsf{BPTIME}(n^d) \subsetneq heur_{1-n^{-a}} - \mathsf{BPP}$.*

1.1 Related Work

Unconditionally, we know very little about probabilistic hierarchies beyond the straightforward observation that $\mathsf{BPTIME}(o(t)) \subsetneq \mathsf{BPTIME}(2^t)$ for time-constructible t, which follows from the hierarchy for deterministic time and the fact that probabilistic time t can be simulated in deterministic time 2^t. The best-known hierarchy [KV87] uses the above result and a translation argument to achieve slightly stronger parameters. However, we still don't know whether there is a language decidable in probabilistic quasi-polynomial time but not in probabilistic linear time. Under plausible complexity-theoretic assumptions, such as that there is a language in E which does not have subexponential size circuits, or that the decision version of the Permanent is not in randomized subexponential time, a strict hierarchy can be shown to exist for probabilistic quasi-polynomial time [IW97, IW01, CNS99]. These complexity-theoretic assumptions are generally believed to be very hard to prove; it is quite possible that a hierarchy theorem for probabilistic polynomial time might be much easier.

Given the lack of success in establishing a hierarchy thus far, it is natural to ask about the status of the question in a relativized setting. $\mathsf{BPP} = \mathsf{P}$ relative to a random oracle [BG81], hence there does exist an oracle relative to which there is a hierarchy. There has been a lot of work towards constructing an oracle relative to which probabilistic polynomial time does not have a hierarchy (see [FS89, FS97, RV01]). Since neither Barak's nor our methods relativize, this work is not directly relevant to the question of whether our methods have the potential to achieve a hierarchy for fully uniform probabilistic polynomial time.

Subsequent to our work, Goldreich, Sudan and Trevisan [GST04] found a method to derive a hierarchy theorem with one bit of advice directly from a hierarchy theorem with $\leqslant \log(n)$ bits of advice under certain conditions. Their method, when used in conjunction with Barak's result, implies Theorem 1.

The organization of the rest of the paper is as follows: First we discuss some technical preliminaries. Then we give a high-level overview of Barak's proof, and sketch the ideas behind our improvements. Finally, we present our modification of Barak's algorithm and our translation argument.

2 Preliminaries

2.1 Complexity Classes

We mostly use standard definitions of complexity classes, which can be found in [BDG88] and [BDG90]. Our definition of probabilistic time-bounded classes is slightly non-standard, however it is equivalent to the standard definition for "nice" time bounds t, such as $t = n^k$ for a fixed constant k. We first define a notion of what it means for a time bound to be "nice".

Definition 3. *A function $t : N \to N$ is said to be time-constructible if there is a deterministic Turing machine M such that on input 1^n, M halts within $O(t(n))$ steps and outputs the value $t(n)$.*

We now proceed to our definition of bounded-error probabilistic time classes.

Definition 4. *Let L be a language and let $t : N \to N$ be a function. We say that $L \in \mathsf{BPTIME}(t)$ if there is a probabilistic Turing machine M such that on all inputs x of length n, with probability $> 2/3$, M halts within $t(n)$ steps on input x and outputs $L(x)$.*

This definition is nonstandard in that the machine is only required to halt within $t(n)$ steps with high probability, rather than with probability 1. This will come in useful in our construction of an optimal algorithm because we will only be able to guarantee that the algorithm halts with high probability within a certain time bound. Note that our definition of probabilistic time classes is equivalent to the standard definition when t is a time-constructible increasing function, as we will be able to use the time-constructibility of t to implement a "timeout" mechanism which converts a machine halting with high probability within time t to a machine that halts with probability 1 within time t.

We also need to clarify the notions of *advice* and *average-case decidability* in the context of probabilistic time classes.

Definition 5. *Given functions $s : N \to N$ and $t : N \to N$ and a language L, $L \in \mathsf{BPTIME}(t)/s$ if there is a probabilistic Turing machine M and a sequence $\{y\}_n, \mid y_n \mid \leqslant s(n)$ of strings such that for all inputs x of length n, with probability $> 2/3$, M halts within $t(n)$ steps on input $< x, y_{|x|} >$ and outputs $L(x)$.*

$\mathsf{BPTIME}(t)/s$ is to be interpreted as the class of languages decided by bounded-error probabilistic machines operating in time t and taking advice of length s. Note that the behavior of an advice taking machine M on input $< x, y >$ where $y \neq y_{|x|}$ may be arbitrary. Our definition is different in this respect from the original definition of Karp and Lipton [KL82], which requires that the advice taking machine has acceptance probability bounded away from 1/2 on all inputs even when the advice is incorrect. If Theorem 1 were to hold under the Karp-Lipton definition of advice-taking probabilistic machines, a hierarchy theorem for uniform probabilistic time would follow immediately.

Definition 6. *Given a function $t : N \to N$ and a language L, $L \in heur_s - \mathsf{BPTIME}(t)$ if there is a probabilistic Turing machine M such that for all input lengths n, for at least a fraction s of inputs x of length n, with probability $> 2/3$, M halts on input x within $t(n)$ steps and outputs $L(x)$.*

$heur_s - \mathsf{BPTIME}(t)$ is to be interpreted as the class of languages decided by bounded-error probabilistic machines in time t on average with parameter s. We use the "heuristic" notion of average-case complexity [Imp95]. The behavior of M on an input not belonging to the "good" set of inputs for which the average-case simulation works may be arbitrary.

2.2 Nice PSPACE-complete language

The proof of our hierarchy theorems proceeds via the construction of an *optimal* algorithm for a PSPACE-complete language L, where an optimal algorithm in this context is a probabilistic algorithm that is at worst polynomially slower than any probabilistic algorithm for L. In order to carry through the proof of our result, we require the PSPACE-complete language L to be *paddable* and *instance-checkable* - we define these two notions below.

Definition 7. *A language L is said to be paddable is there is a deterministic polynomial-time computable procedure P which given inputs x of length n and 1^m where $m \geqslant n$, outputs a string of length m such that $P(x, 1^m) \in L$ iff $x \in L$.*

Intuitively, a language is paddable, if given a string x, it is possible to efficiently generate arbitrarily long strings which behave the same as x as far as membership in the language L is concerned.

Definition 8. *[BK95] A language L is said to be instance-checkable if there is a probabilistic polynomial-time oracle procedure I which, given an oracle P and an input x of length n, has the following properties:*

1. *I outputs one of 3 values "0", "1" or "?" at the end of each computation path.*

2. *If P decides L correctly, I outputs $L(x)$ with probability 1.*

3. I outputs $1 - L(x)$ with probability at most $2^{-\Omega(n)}$, irrespective of the oracle P.

Intuitively, an instance checker for a language L tests whether a program P decides L correctly on an input x or not. It follows from the proof of the $\mathsf{IP} = \mathsf{PSPACE}$ [LFKN92, Sha92] result that all PSPACE-complete problems have instance checkers, and it follows from the proof of the $\mathsf{MIP} = \mathsf{NEXP}$ [BFL91] that all EXP-complete languages have instance checkers.

Thus far, the properties we require of the PSPACE-complete language L are similar to the properties required by Barak of the EXP-complete language for which he constructs an optimal algorithm. Indeed, the proof of Theorem 1 only uses these properties. For the proof of Theorem 2, we require an additional property, namely that L be worst-case to average-case reducible. Informally, this means that the existence of an machine M which for each n decides L correctly with high probability over the uniform distribution on inputs of length n implies the existence of a machine M' which is at most polynomially slower than M and decides L correctly on all inputs.

Building on ideas of Trevisan and Vadhan [TV02], we define a PSPACE-complete language satisfying slightly stronger forms of the properties mentioned above, which are useful in the construction of our optimal algorithm and in our proofs of correctness:

Theorem 9. *There is a PSPACE-complete language $L \in \mathsf{DTIME}(2^{2n})$ and a linear-time decidable set $S \subset \{0,1\}^*$ with the following properties:*

1. *(Paddability)*
   - *(a) There is a deterministic linear-time procedure P, which given x of length n and 1^m as inputs, where $m \geqslant n$, produces a string $P(x, 1^m)$ of length n in $\overline{S}$ such that $x \in L$ iff $P(x, 1^m) \in L$.*
   - *(b) There is a deterministic linear-time procedure P_{rev}, which given a string $x \in \overline{S}$ of length n as input, produces a string $P_{rev}(x) \in S$ of length $< n$ such that $x \in L$ iff $P_{rev}(x) \in L$.*
2. *(Instance-Checkability) L is instance-checkable with a checker I that only makes queries to the oracle about strings of length n on any input x of length n.*
3. *(Worst-case to average-case reducibility)*
   - *(a) $|S \cap \{0,1\}^n| = 2^{n-1}$.*
   - *(b) There is a polynomial p and polynomial-time procedures q and f such that for each integer n, for each input $x \in S$ of length n, for each integer $1 \leqslant i \leqslant p(n)$ the distribution of $q(i, x, r)$ over coin tosses r is uniform over $S \cap \{0,1\}^n$ and $L(x) = f(x, r, L(q(1, x, r)), L(q(2, x, r)) \ldots$
 $L(q(p(n), x, r)))$ with probability at least $1 - 2^{-\Omega(n)}$ over coin tosses r.*

Proof. It is implicit in the work of Trevisan and Vadhan [TV02] that there is a PSPACE-complete language $L' \in \mathsf{DTIME}(2^{2n})$ with the following properties:

1. L' has a worst-case to average-case reduction, with the queries at length n uniformly distributed over $\{0,1\}^n$.

2. L' is instance-checkable with an instance checker I' which, on input of length n, only makes queries to its oracle about strings of length at most n.

From L', we define a language L with the stated properties. L is essentially L' "with padding". More precisely, for any integer $i \geqslant 0$, $0^i 1x \in L'$ iff $x \in L$.

It is not hard to see that Property 1 in the statement of Theorem 9 holds for L'. The set S in the statement of the theorem can be taken to be $\{0^i 1x | i \geqslant 1\}$. Given an x of length n such that x is of the form $0^j 1x'$, The procedure $P(x, 1^m)$ simply outputs $0^{j+m-n} 1x'$. Similarly, given a string of the form $0^j 1x'$ as input where $j \geqslant 1$, P_{rev} outputs $1x'$.

We can also define an instance checker I for L given the instance checker I' for L' such that I only makes queries of the same length as its input. Whenever I' makes a query to its oracle, I pads the query up to its input length using the procedure P before asking the query to the oracle. We also need to ensure that for heavily padded inputs, the probability of outputting the wrong answer is exponentially small, but this can be done by just repeating the simulation of I' a number of times depending on how heavily padded the input is.

As for the average-case to worst-case connection, uniform distribution of queries over $\{0,1\}^n$ for L' translates directly to uniform distribution of queries over S for inputs in S for L. □

Intuitively, procedure P in the statement of Theorem 9 is a padding procedure, and procedure P_{rev} is a "reverse padding" procedure, which when given a padded input, recovers the original input. S is the set of non-padded inputs. Half the inputs of any given length belong to this set, and the worst-case to average-case reduction works for inputs in this set.

3 Overview of techniques

3.1 Barak's proof

Barak's proof represents a completely new approach towards proving a hierarchy theorem for probabilistic polynomial time. We sketch the ideas behind his proof here.

The critical idea of Barak is to find a language L with an *optimal* algorithm. An optimal algorithm is a probabilistic algorithm for L that is only polynomially slower than the "best" probabilistic algorithm for L. To be more precise, there is a constant c such that if there is a probabilistic algorithm solving L running in time t, then the optimal algorithm runs in time $O(t^c)$. Thus if the optimal algorithm runs in time T, $L \in \mathsf{BPTIME}(T) - \mathsf{BPTIME}(T^{1/c'})$ for all $c' > c$. If T were a constructible time bound, we could use a translation argument to show that an appropriately padded version L' of L is in $\mathsf{BPTIME}(n^{c'})$ but not in $\mathsf{BPTIME}(n)$, which implies a hierarchy theorem for probabilistic polynomial time.

The question of how to find a language L with an optimal algorithm still needs to be addressed. Barak observes that any EXP-complete language has an optimal algorithm, indeed any language with an *instance checker* has an optimal algorithm, and it follows from the rich theory of probabilistically checkable proofs that all EXP-complete languages have instance checkers.

Let $P_1, P_2 \ldots$ be an effective enumeration of all probabilistic Turing machines (where there is no guarantee that a machine in the enumeration is bounded-error). Fix an EXP-

complete language L and let I be an instance checker for L. Barak defines an optimal algorithm A for L as follows: Given an input x, A successively runs I with oracles $P_1, P_2 \ldots P_{f(n)}$ on x, where f is some polynomially-bounded easily computable function. Whenever I asks an oracle query to P_i, A simulates P_i to determine the answer. If I^{P_i} returns a Boolean value for some i, A outputs that value. If I^{P_i} returns "?", A continues the simulation with the next program in the enumeration. We have skirted the issue of how long A simulates the machines P_i - Barak's algorithm actually runs in stages $m = 1, 2 \ldots$, and at a stage m, oracle queries to a program P_i are simulated for m steps.

Given the properties of an instance checker, it is not hard to show that A is an optimal algorithm for L. If P_e is a probabilistic bounded-error algorithm for L which halts with high probability after t steps, then for large enough n (i.e., for n such that $f(n) > e$), P_e will be one of the oracles that A uses, and hence at stage $m = t$, A will halt with the correct answer with high probability after simulating I with oracle P_e. A has used at most $t^2 poly(n)$ time up to this stage, which is still polynomial in t. Any incorrect program Q can only contribute an exponentially small error when I is run with Q as oracle[1], hence A solves L correctly and halts within $poly(t)$ steps with high probability.

As discussed before, an optimal algorithm implies a hierarchy theorem if the time bound of the optimal algorithm is time-constructible. However, it seems to be hard to argue that this is the case. Barak's solution is to use an advice string of size $\log(\log(n))$ to represent an approximation to the time taken by the optimal algorithm, in which case the translation argument can be pushed through. We suggest more efficient solutions to this problem.

3.2 Our improvements

We get our first improvement over Barak using a translation argument efficient with respect to advice. We require that the pads conform to a specific format, which ensures that for a given length m, there is at most one input length n such that inputs of length n can be padded to length m. If this holds, then just one bit of advice is required to tell whether m is a "good" length, meaning that the simulation of the optimal algorithm on inputs of length n takes $O(poly(m))$ time. Of course, we also need to take care that the constraint we impose on the pads doesn't make the simulation of the optimal algorithm infeasible, but this isn't hard.

We do obtain a hierarchy theorem for probabilistic polynomial time with one bit of advice using this translation argument. However, the value of the advice bit depends on the time taken by the optimal algorithm, and it is not clear at all whether this function is easily computable. For example, it might be the case that the optimal algorithm takes vastly different times for different inputs of the same length. In such a case, it may not be easy to estimate the worst-case time on a given input length. We do have some flexibility, though, in designing our algorithm. It is well-known that there are PSPACE-complete problems[2] with

[1] A subtle but critical point is that we are now considering the probability over coin tosses of Q as well, and Q may not be bounded-error. However we may interpret Q as a probability distribution over oracles each of which causes I to output a wrong answer with exponentially small probability, and hence this property is preserved when we take into account the coin tosses of Q as well.

[2] For technical reasons, we use a PSPACE-complete language rather than an EXP-complete language as in

a worst-case to average-case connection, i.e., they are just as hard to solve in the average case as in the worst case. This suggests that the average-case running time of an optimal algorithm for a PSPACE-complete problem may be polynomially related to its worst-case running time. We do not know if Barak's algorithm has this property but we are able to modify his optimal algorithm and impose some structure on it so that we obtain a version of the worst-case to average-case connection. Specifically, we are able to design a probabilistic polynomial-time procedure that produces a rough estimate of the worst-case running time from estimates of a different but related quantity for random inputs.

However, it is still not clear whether this gains us anything since the estimate we obtain is quite rough, and is moreover obtained probabilistically. We need another idea, which involves the use of the randomness in the input itself to disambiguate the computation and maintain the bounded-error promise with high probability in the case when the estimate is too close to the length of the pad. Using the estimation procedure and this idea in conjunction with our translation argument, we derive a hierarchy theorem for average-case probabilistic polynomial time if the optimal algorithm does not run in polynomial time.

If the optimal algorithm runs in polynomial time, the fact that our problem is PSPACE-complete implies that we can use a diagonalization technique to directly obtain a hierarchy for average-case probabilistic polynomial time. Thus an average-case hierarchy theorem holds for probabilistic polynomial time unconditionally.

4 Our Results

4.1 An Optimal Algorithm

Let L be a PSPACE-complete language as in the statement of Theorem 9. Let I be an instance checker for L and let q and f be polynomial-time procedures implementing a worst-case to average-case reduction for L, such that I and f both have error probability at most 2^{-4n} on inputs of length n. We describe an optimal algorithm for L. The algorithm runs in stages $m = 1, 2 \ldots$. At stage m, for each machine M of description length $3 \log\log(m)$ (note that there are at most $(\log(m))^3$ such machines), we check to see if M^m, i.e., M restricted to m steps, is a good candidate for deciding L by running the worst-case to average-case reduction for L on top of the instance checker. If M^m were indeed a good candidate, the instance checker I would return a non-"?" value with high probability on all instances of length n and the worst-case to average-case reduction would give a correct answer with high probability as well. If M^m is not a good candidate, we know by the properties of I that a wrong answer is produced with very small probability, hence the bad candidate cannot affect the performance of the algorithm by very much.

We describe the algorithm in Table 1.

For each n, let $T(n)$ be the minimum number t such that for each $i, 1 \leqslant i \leqslant n$ and for each input of length i, with probability at least $1 - 1/i$, Algorithm OPTIMAL halts and produces the correct answer within t steps. Then $L \in \mathsf{BPTIME}(T(n))$. In order for the definition of $T(n)$ to make sense, we need to ensure that such a number t exists. For any $x \in S$ of length

[Bar02].

```
Algorithm OPTIMAL:
Input: String $x$ of length $n$

1. If $x \notin S$, $x \leftarrow P_{rev}(x)$
2. If $|x| \leqslant \log(n)$, run the natural deterministic exponential time
   algorithm for $L$ on $x$ and exit.
3. For $m = 1, 2 \ldots$ do:
4. For each probabilistic Turing machine $M$ of size $3 \log\log(m)$ do:
   (a) For $j = 1 \ldots \log(n)$ do:
       i. Run $q$ on $x$ with randomly chosen $r$ to generate queries
          $q(1, x, r), q(2, x, r) \ldots q(|x|^k, x, r)$.
       ii. For $i = 1 \ldots |x|^k$ do:
           A. Run $I$ on $q(i, x, r)$ with oracle $M^m$, and if an answer
              $val$ is returned, $guess(q(i, x, r)) \leftarrow val$, otherwise
              $guess(i, x, r)$ is undefined.
       iii. If $guess(i, x, r)$ is defined for all $i, 1 \leqslant i \leqslant |x|^k$, set
            $f_j \leftarrow f(x, r, guess(q(1, x, r)) \ldots guess(q(|x|^k, x, r)))$
   (b) If $f_j$ is defined for all $1 \leqslant j \leqslant \log(n)$, output the majority
       value of the $f_j$'s and exit all loops, else continue.
```

Table 1: Optimal Algorithm

i, such a number t exists because a deterministic linear exponential-time machine deciding L will eventually be tried by Algorithm OPTIMAL and OPTIMAL will return an answer with high probability when it tries this machine. Moreover, the probability that a wrong answer is output before this stage is exponentially small, since both the instance checker I and the worst-case to average-case reduction f produce wrong answers with exponentially small probability. Any $x \notin S$ of length i is transformed to $x' = P_{rev}(x)$ of length $i' < i$. Now the same argument as for $x \in S$ works and the fact that the worst-case to average-case reduction is run $\log(i)$ times independently ensures that the probability of halting is at least $1 - 1/i$. Indeed, since $L \in \mathsf{DTIME}(2^{2n})$, this argument yields an explicit upper bound of 2^{6n} for T.

Now, we will consider two cases. The first is that $T(n)$ is polynomially bounded. We will dispose of this case with a diagonalization argument. The other case is that $T(n)$ is superpolynomial infinitely often. In this case, firstly we will show that OPTIMAL is an optimal algorithm and hence there is a constant $\epsilon > 0$ such that $L \in \mathsf{BPTIME}(T) - \mathsf{BPTIME}(T^{\epsilon})/2\log\log(T)$. Next, we will show how to define, for each constant d, a padded version of L that is decidable in BPP with just one bit of advice but is not in $\mathsf{BPTIME}(n^d/\log\log(n))$. This will yield our Theorem 1.

4.2 The case $L \in \mathsf{BPP}$

First, note that by the PSPACE-completeness of L, if L can be decided in BPP, so can every language in PSPACE.

Proposition 10. *If* $L \in \mathsf{BPP}$*, then* $\mathsf{PSPACE} = \mathsf{BPP}$.

Next, we argue that if $\mathsf{PSPACE} = \mathsf{BPP}$, we obtain hierarchy theorems for probabilistic polynomial time with one bit of advice and for average-case probabilistic polynomial time. These arguments use the technique of diagonalization. The first argument is a straightforward diagonalization, where we observe that a "universal" language L_u for probabilistic polynomial-time machines is in PSPACE and that we can diagonalize in PSPACE, and hence by assumption in BPP, against this language.

Lemma 11. *Let* $d \geqslant 1$ *be a constant. If* $\mathsf{PSPACE} = \mathsf{BPP}$*, then* $\mathsf{BPP} \not\subseteq \mathsf{BPTIME}(n^d)/\log(n)$.

Proof. We assume an enumeration of probabilistic machines for which the description length of a machine M can be padded to any larger input length - it is not hard to guarantee this property. Now we define L_u as follows: the code $< M >$ of a machine M belongs to L_u iff M accepts with probability $< 1/2$ when run for $|< M >|^{d+1}$ steps on input $< M >$. Given the paddability of description lengths, it is clear that for each language $L' \in \mathsf{BPTIME}(n^d)/\log(n)$, for large enough n, there is an input (namely, the code of the BPTIME machine together with the advice padded to length n) on which L_u differs from L'. Thus $L_u \notin i.o.\mathsf{BPTIME}(n^d)/\log(n)$. On the other hand, it is not hard to see that $L_u \in \mathsf{PSPACE}$, and hence by assumption, $L_u \in \mathsf{BPP}$. □

The second diagonalization argument, due to Wilber [Wil83] gives an average-case hierarchy for BPP under the assumption $\mathsf{PSPACE} = \mathsf{BPP}$.

Lemma 12. *Let* $d \geqslant 1$ *be a constant. If* $\mathsf{PSPACE} = \mathsf{BPP}$*, then* $\mathsf{BPP} \not\subseteq heur_{2/3}{-}\mathsf{BPTIME}(n^d)$.

Proof. It is known [Wil83] that for every $d > 1$, there is a language L_d such that $L_d \in \mathsf{DSPACE}(n^{d+1})$ and $L_d \notin heur_{2/3} - \mathsf{DSPACE}(n^d)$. By assumption, $L_d \in \mathsf{BPP}$ and since $heur_{2/3} - \mathsf{BPTIME}(n^d) \subseteq heur_{2/3} - \mathsf{DSPACE}(n^d)$, $L_d \notin heur_{2/3} - \mathsf{DSPACE}(n^d)$, proving the theorem. □

4.3 Proof of Optimality

We need to show that Algorithm OPTIMAL is no worse than polynomially slower than any probabilistic algorithm for L. The idea behind the proof is that if M is a probabilistic algorithm for L which with high probability halts and outputs the correct answer within t steps, then the instance checker I when run on oracle M^t would output the correct value for an instance with high probability, and the worst-case to average-case reduction would also work with high probability. Thus, the time required for A to output an answer with high probability is bounded above by the time required to get to the simulation of M^t in stage t, which is bounded above by some polynomial in t.

We actually need to show that Algorithm OPTIMAL is no worse than polynomially slower than any probabilistic algorithm taking a small amount of advice, but this does not introduce too many additional complications.

Lemma 13. *If $L \notin \mathsf{BPP}$, there is a constant $\epsilon > 0$ such that for each constant $b > 0$, $L \notin \mathsf{BPTIME}(n^b + T^\epsilon)/2\log\log(T)$.*

Proof. Fix b. We pick $\epsilon = 1/4$. Let M be a probabilistic machine of description size $2\log\log(T) + O(1)$ which, for each input x of length n, with probability at least $1 - 1/n$ halts and outputs the correct answer $L(x)$ within $T^\epsilon + n^b$ steps. We derive a contradiction for our value of ϵ.

Since $L \notin \mathsf{BPP}$, there is an infinite set J of input lengths such that for all $n \in J, T(n) \geqslant n^{b+\alpha} + n^{\alpha/(1-2\epsilon)}$, where $\alpha > 0$ is a constant to be specified later. We shall show that M cannot satisfy our assumption for inputs with input length in J. Consider an input x of length $n \in J$. We can assume without loss of generality that $x \in S$ (otherwise we use the same argument with $x \leftarrow P_{rev}(x)$). By running the machine M $2^{\lceil 2\log(n)\rceil}$ times independently and taking the majority vote, we can define a machine M_1 with description length $2\log\log(T) + \log\log(n) + O(1) \leqslant 3\log\log(T^\epsilon)$ which within $T^\epsilon n^2 + n^{b+2}$ steps, halts and outputs the correct answer with high probability. Thus, when Algorithm OPTIMAL reaches stage $T^\epsilon n^2 + n^{b+2}$ and runs the worst-case to average-case reduction on top of the instance-checking routine I with oracle $M_1^{T^\epsilon n^2 + n^{b+2}}$, it halts and outputs the correct answer with probability at least $1 - 2^{-3n}$(since the probability of I outputting a wrong value or "?" on any of the n^k instances of length n queried by I and the probability that the function f will return the wrong value are both at most 2^{-4n}). Also, the probability that Algorithm OPTIMAL has output a wrong value before this stage is at most 2^{-2n}, hence we have that Algorithm OPTIMAL halts and outputs the correct answer with probability at least $1 - 2^{-n} > 1 - 1/n$ within time $(T^{2\epsilon} + n^b)n^\alpha$ for some constant α. We have a contradiction since $(T^{2\epsilon} + n^b)n^\alpha < T$ if $n \in J$. □

4.4 Proof of Theorem 1

In this section, we prove Theorem 1. We define a padded version L_{pad} of the language L, where the pad represents the running time of algorithm OPTIMAL on the underlying input, such that $L_{pad} \in \mathsf{BPP}/1$ but $L_{pad} \notin \mathsf{BPTIME}(n^d)/\log\log(n)$. Let ϵ be a constant as in the statement of Lemma 13.

$L_{pad} = \{x\#1^y | y = 2^{2^z}$ for some $z, x \in L, y > |x|, y + |x| + 1 \geqslant T(|x|)^{\epsilon/3d}\}$

First, we show $L_{pad} \in \mathsf{BPP}/1$. The basic idea is that by requiring the pads to conform to a specified structure, we make do with one bit of advice to tell us whether the length of the pad is sufficient to carry out the simulation of Algorithm OPTIMAL.

Lemma 14. $L_{pad} \in \mathsf{BPTIME}(n^{3d/\epsilon})/1$

Proof. We construct a probabilistic polynomial-time machine M operating in time $n^{3d/\epsilon}$ and taking one bit of advice which accepts L_{pad}. The basic idea is that given an input of the form $x\#1^y$ with $|x| + 1 + y = m$, M first checks if the input has a valid form. This can be done in quasilinear time. If the input does not have a valid form, M rejects. Otherwise, let b_m be the advice bit for M at length m. M accepts iff $b_m = 1$ and Algorithm OPTIMAL halts and accepts on x within $n^{3d/\epsilon}$ time steps.

First we specify what sequence of advice bits $b_i, i = 1 \ldots \infty$ M receives and then we show that for any m and for any x' of length m, $x' \in L_{pad}$ iff M accepts x' with advice b_m. Consider any input length m. Call m "good" if $m = n_m + y_m + 1$ for some $n_m, y_m > 0$ such that $y_m = 2^{2^{z_m}}$ for some $z_m \geqslant 0$ and $y_m > n_m + 1$. Thus $y_m > m/2$. If m is good, then n_m and y_m are determined uniquely. For suppose not, and let $n'_m \neq n_m$ and y'_m be such that $n'_m + y'_m = n_m + y_m$, $y'_m = 2^{2^{z'_m}}$ for some $z'_m \geqslant 0$ and $y'_m \geqslant n'_m$. Without loss of generality $y'_m > y_m$. We have that $m > y'_m \geqslant y_m^2 \geqslant m^2/4$, which is a contradiction for $m > 4$.

Let $T'(n_m) = T(n_m)^{\epsilon/3d}$. Now we specify the advice bits. If m is not good, $b_m = 0$. If m is good, $b_m = 1$ iff $m \geqslant T'(n_m)$.

Next we fix an m and show that for x' of length m, $x' \in L_{pad}$ iff M accepts x'. The forward direction follows immediately from the specification of the advice bits. For the reverse direction, we show that if $x' \notin L_{pad}$ for input x' of length m, then M halts within the required time bound and rejects x' with probability close to 1. If x' is not of the form $x\#1^y$ for some y of the form 2^{2^z} such that $y > |x|$, M rejects with probability 1 since the deterministic test that the input has a valid form fails. Otherwise, let x' be of the form $x\#1^{y_m}$, where x is of length n, $y_m = 2^{2^{z_m}}$ for some integer $z_m \geqslant 0$, and $y > |x|$. There are two cases: (1) $T'(n) > m$, and (2) $T'(n) \leqslant m$. In case (1), the advice bit b is 0 and hence M rejects with probability 1. In case (2), M has enough time to simulate Algorithm OPTIMAL on x and hence rejects with high probability when $x \notin L$. □

Next, we show $L_{pad} \notin \mathsf{BPTIME}(n^d)/\log\log(n)$ if the optimal algorithm for L does not run in polynomial time. The idea here is that if $L_{pad} \in \mathsf{BPTIME}(n^d)/\log\log(n)$ via an advice-taking probabilistic machine M, we can solve L in time $T^\epsilon + poly(n)$ with a small amount of advice. The advice is interpreted as coming in two parts. The first part of the advice suggests a "good" input length m to which we pad the input, and we then simulate M on the padded input using the second part of the advice as advice to M. The total amount of advice used is $< 2\log\log(T)$, and thus we obtain a contradiction to Lemma 13.

Lemma 15. *If $L \notin \mathsf{BPP}$, $L_{pad} \notin \mathsf{BPTIME}(n^d)/\log\log(n)$.*

Proof. The proof is by contradiction. Assume $L_{pad} \in \mathsf{BPTIME}(n^d)/\log\log(n)$. Let M be an advice-taking probabilistic machine running in time n^d and accepting L_{pad}. We construct an advice-taking probabilistic machine M' running in time $n^{3d} + T(n)^\epsilon$ with $2\log\log(T)$ bits of advice and accepting L, which is a contradiction to Lemma 13.

M' acts as follows: given an input x of length n, it interprets the first $\log\log(n)$ bits of its advice as an encoding of the smallest z such that $2^{2^z} > n$ and $2^{2^z} + n + 1 \geqslant T(n)^{\epsilon/3d}$. It then pads its input x with $1^{2^{2^z}}$ and runs M on the padded input $x' = x\#1^{2^{2^z}}$, interpreting the second part of its advice as the correct advice string for M on the padded length. It outputs the result of the simulation of M on x'. Note that $|x'| \leqslant max(n^3, T^\epsilon)$. Thus M' uses time at most $n^{3d} + T(n)^\epsilon$ on x. Also, if both parts of the advice string are correct, the simulation has enough time to run, and hence M' decides correctly whether x is in L. □

If $L \in \mathsf{BPP}$, Proposition 10 and Lemma 11 imply Theorem 1. If $L \notin \mathsf{BPP}$, Lemma 14 and Lemma 15 together imply Theorem 1. Thus Theorem 1 holds unconditionally. The existence of a hierarchy theorem for probabilistic polynomial time with one bit of advice follows from Theorem 1 by a standard translation argument.

4.5 Estimation of Running Time

In this subsection, we show that there is an efficient procedure estimating the running time of Algorithm OPTIMAL to within a polynomial factor with high probability. This is where we reap the advantage of running a worst-case to average-case reduction on top of the instance checking. Intuitively, we can get a rough estimate of the time required to decide L by estimating the probability that the instance checker I returns a non-"?" value. The worst-case to average-case reduction ensures that if for some machine M there is some input x of length n such that there is a significant (i.e., at least inverse polynomial) probability of not getting an answer for x within time t after running I with $M^{t/poly(n)}$ as oracle and then running the worst-case to average-case reduction on top, then for at least a polynomial fraction of the inputs of length n, there is a significant chance that the instance checker will return a "?" value when it is given $M^{t/poly(n)}$ as oracle.

The proof of correctness of our estimation procedure requires standard Chernoff bounds ([AS92]) as stated below:

Proposition 16. *Let $Y_1 \dots Y_n$ be independent random variables with $Pr(Y_i = 1) = p$ and $Pr(Y_i = 0) = 1 - p$ for each i. Let $Y = \sum_{i=1}^{n} Y_i$. Then $Pr(Y > np + a) < e^{-2a^2/n}$ and $Pr(Y < np - a) < e^{-a^2/2pn}$.*

Lemma 17. *There is a probabilistic procedure ESTIMATE and constants a and c such that ESTIMATE, when given an integer n as input, with probability at least $1-1/2^{n^2}$, halts within $T(n)n^c$ steps and outputs a number T_{est} such that $T(n)/n^a \leqslant T_{est} \leqslant T(n)n^a$. Moreover, there is at most one number T_{mid} such that $|Pr(T_{est} < T_{mid}) - Pr(T_{est} \geqslant T_{mid})| < 1/4Tn^a$.*

Proof. We define some notation that will be used in the proof of correctness of the procedure for estimation of the running time. Fix an input length n. An *execution* is a pair (m, M) consisting of a running time m and a machine M of description length $\leqslant 3\log\log(m)$. An input x of length n is said to be *(m,M)-good* if the probability that I^{M^m} returns a "?" value on x is at most $1/n^{k+2}$. The execution (m, M) is said to be *n-good* if at least a fraction $1 - 1/(n^{k+2})$ of the inputs of length n are (m, M)-good.

Let the running times of procedures I, q and f in Algorithm OPTIMAL be bounded by n^{r_1}, n^{r_2} and n^{r_3} respectively. Let $a = 3(k + r_1 + r_2 + e)$, and $c = 16k + 3r_1 + 3r_2 + 8$.

The procedure ESTIMATE works as follows: It runs the following process $2n^2$ times independently. In the rth run, $1 \leqslant r \leqslant 2n^2$, it first sets m = 2. For each machine M of description length $\leqslant 3\log\log(m)$, for each $\log(n) \leqslant i \leqslant n$, it estimates whether the execution (m, M) is i-good by generating i^{8k+2} random strings $x_1 \dots x_{i^{8k+2}}$ of length i and estimating whether at least $i^{8k+2} - i^{7k}/2$ of these strings are (m, M)-good. For estimating whether a string x_j of length i is (m, M)-good, it runs I^{M^m} on x_j i^{8k+2} times independently and checks whether at least $i^{8k+2} - i^{7k}/2$ of these runs yield a value that is not "?". If there is an M of description length $\leqslant 3\log\log(m)$ such that (m, M) is estimated to be i-good for all $\log(n) \leqslant i \leqslant n$, ESTIMATE sets $t_r \leftarrow mn^{a/2}$ and begins the $r + 1$th run. If there is no such M, ESTIMATE sets $m \leftarrow 2m$ and repeats the estimation process within the rth run.

Let T_1 be the median of the values $t_r, r = 1 \leqslant 2n^2$. ESTIMATE outputs T_{est}, where T_{est} is an integer chosen uniformly at random in $[T_1 n{-}2a/3, T_1 n^{2a/3}]$

We need to show that with high probability, ESTIMATE halts within $T(n)n^c$ steps and neither overestimates nor underestimates $T(n)$ by more than a n^a factor. First we show that for each $r, 1 \leqslant r \leqslant 2n^2$, with probability at least $1 - 1/2n$, $t_r \leqslant T(n)n^{a/3}$. This implies that with probability at least $1 - 2^{n^2+1}$, $T_1 \leqslant T(n)n^{a/3}$ which implies $T_{est} \leqslant T(n)n^a$. Fix r such that $1 \leqslant r \leqslant 2n^2$. Let M be a machine implementing algorithm OPTIMAL. Define a modified version M_n of M which runs M $2^{\lceil 2\log(n)\rceil}$ times independently for $2^{\lceil \log(T(n)\rceil)}$ steps and outputs the majority value of these runs if at least one of these runs produces a value, otherwise it outputs an arbitrary value. The description length of M_n is at most $|M| + \log\log(T(n)) + \log\log(n) + O(1)$, which is at most $3\log\log(T(n))$, since $T(n) \geqslant n$. By applying the bounds of Proposition 16, we see that the success probability of M_n on any string of length at most n is at least $1 - 2^{-n}$, and hence the probability that $I^{M_n^{2^{\lceil T(n)\rceil + \lceil 2\log(n)\rceil}}}$ produces a non-"?" value is at least 2^{-n+1}. Again by applying the bounds of Proposition 16, for any given $i \geqslant \log(n)$ the probability that the execution $X = (2^{\lceil \log(T(n))\rceil + \lceil 2\log(n)\rceil}, M_n)$ passes all the tests of the rth run of procedure ESTIMATE is at least $1 - 1/n^3$. By a union bound, the execution X passes all the tests for all $\log(n) \leqslant i \leqslant n$ with probability at least $1 - 1/n^2$. Thus $t_r \leqslant 2T(n)n^{r_1+k} + n^{r_2+k} + n^e \leqslant T(n)n^{r_1+r_2+e+k} = T(n)n^{a/3}$ with probability at least $1 - 1/n^2$. Also note that the time spent by procedure ESTIMATE in the rth run until it finishes all tests for execution X is at most $T(n)n^{16k+r_1+r_2+e+5}$, and thus the total time for all runs is at most $T(n)n^c$.

The argument that ESTIMATE does not underestimate $T(n)$ is more complicated. Again fix an r such that $1 \leqslant r \leqslant 2n^2$. We show that with probability at least $1 - 1/2n$, $t_r \geqslant T(n)n^{-a/3}$, which implies $T_1 \geqslant T(n)n^{-a/3}$ with probability at least $1 - 1/2^{n^2+1}$ and hence $T_{est} \geqslant T(n)n^{-a}$ with at least that probability. Define $l(n) = T(n)/n^{a/3}$. Assume for the purpose of contradiction that with probability greater than $1/2n$, $t_r \leqslant l(n) = T(n)/n^{r_1+r_2+e+k}$. There are at most n^4 executions (m, M) tried by the procedure ESTIMATE in the rth run(since only values of m which are powers of 2 are tried, and for each such value m, at most $(\log(m))^3$ machines M are considered), hence there is an execution (m, M) such that $m < l(n)$ and (m, M) is i-good for all $\log(n) \leqslant i \leqslant n$ with probability at least $1/2n^5$ over the coin tosses of ESTIMATE. Fix such an execution (m, M).

We first show that the probability that an input of length i is (m, M)-good is at least $1 - 1/i^{k+2}$ for each $\log(n) \leqslant i \leqslant n$. Assume, to the contrary, that this probability is less than $1 - 1/i^{k+2}$. We show that then the probability that ESTIMATE estimates (m, M) to be i-good is less than $1/2n^5$. By the Chernoff bounds of Proposition 16, the probability that a string of length i is estimated to be (m, M)-good if it is not (m, M)-good is at most $1/n^6$ if $i \geqslant \log(n)$. Again by Proposition 16, the probability that (m, M) is estimated to be i-good when the fraction of strings of length i that are (m, M)-good is less than $1 - 1/i^{k+2}$ is at most $1/n6$. Hence the probability that ESTIMATE estimates (m, M) to be i-good is at most $2/n^6 < 1/2n^5$ for n large enough , which is a contradiction.

Let $m' = mn^{r_1+k} + n^{r_2+k} + n^e$. Next we show that if the probability that an input of length i is (m, M)-good is at least $1 - 1/i^{k+2}$ for each $\log(n) \leqslant i \leqslant n$, then $T(n) \leqslant m'$, which is a contradiction if $m < l(n)$. We need to show that under the assumption, Algorithm OPTIMAL takes at most time m' on every input of length at most n. Let x of length i be the input of length $\leqslant n$ on which Algorithm OPTIMAL takes maximum time. It cannot be the case that $i < \log(n)$, since Algorithm OPTIMAL takes at most time n on inputs of length $\leqslant \log(n)$. We can assume without loss of generality that $x \in S$ (since if $x \notin S$, Algorithm OPTIMAL runs instead on an equivalent input in S which is computed from x in linear time). Since $S \cap \{0,1\}^i = 2^{i-1}$ by Theorem 9, the probability that an input of length i in S is (m, M)-good is at least $1 - 1/2i^{k+1}$. Consider the behavior of algorithm OPTIMAL during execution (m, M) on input x. It follows from the form of the average-case to worst-case connection for L stated in Theorem 9 that queries which are not (m, M)-good appear on at most a fraction $1/2i$ of computation paths. For computation paths on which only (m, M)-good queries appear, with probability at least $1 - 1/2i$ over the coin tosses of I, the correct answer is output within time $mi^{r_1+k} + i^{r_2+k} + i^e \leqslant m'$ since $i \leqslant n$. Thus with probability at least $1 - 1/i$, Algorithm OPTIMAL halts and outputs the correct answer on x, which implies $T(n) \leqslant m'$, and hence a contradiction.

Now that we have established that ESTIMATE outputs a good approximation to T with high probability, we need to show that the distribution of values output by ESTIMATE is "smooth". The smoothness property we require, roughly speaking, is that the distribution has at most one median - this will come in useful in our proof of Theorem 2.

Let T_{mid} be the least number such that $|Pr(T_{est} < T_{mid}) - Pr(T_{est} \geqslant T_{mid})| \leqslant 1/4Tn^a$. We show that in this case, $Pr(T_{est} = T_{mid}) > 1/4Tn^a$, which implies that there is no other number T' such that $|Pr(T_{est} < T' - Pr(T_{est} \geqslant T')| \leqslant 1/4Tn^a$.

Note that the fact that ESTIMATE is a good approximator of T with high probability implies that $T_{mid} \in [T/n^a, Tn^a]$. We consider 3 cases - (1) $T_{mid} \in [T/n^{a/3}, Tn^{a/3}]$, (2) $T_{mid} \in [Tn^{a/3}, Tn^a]$, and (3) $T_{mid} \in [T/n^a, T/n^{a/3}]$. In case (1), $Pr(T_1 \in [T/n^{a/3}, Tn^{a/3}] \geqslant 1 - 1/2^{n^2}$ as established before, and $Pr(T_{est} = T_{mid} | T_1 \in [T/n^{a/3}, Tn^{a/3}]) \geqslant 1/Tn^a$ by the definition of ESTIMATE and the assumption on the range of T_{mid}, hence $Pr(T = T_{mid}) \geqslant 1/2Tn^a$.

Now we turn to case (2), and the proof for case (3) will follow by a symmetrical argument. $Pr(T_{est} \geqslant T_{mid}) = Pr(T_{est} \geqslant T_{mid} | T_1 \geqslant Tn^{a/3}) Pr(T_1 \geqslant Tn^{a/3}) + Pr(T_{est} \geqslant T_{mid} | T_1 < Tn^{a/3}) Pr(T_1 < Tn^{a/3}) \leqslant 1/2^{n^2} + Pr(T_{est} = T_{mid}) Tn^a$, which implies $Pr(T_{est} = T_{mid}) > (1/2 - 1/2^{n^2}) 1/Tn^a$, and thus $Pr(T_{est} = T_{mid}) > 1/4Tn^a$ when $n > 4$. □

4.6 Proof of Theorem 2

In this section, we outline a proof of Theorem 2. The proof relies on the ability to estimate the running time of Algorithm OPTIMAL on inputs of length n, as guaranteed by Lemma 17, as well as the translation argument with pads of a fixed structure utilised in the proof of Theorem 1.

As in the proof of Theorem 1, we consider 2 cases. In the case that $L \in \mathsf{BPP}$, Proposition 10 and Lemma 12 imply Theorem 2 .

Lemma 18. *If $L \in \mathsf{BPP}$, then $\mathsf{BPP} \not\subseteq heur_{2/3} - \mathsf{BPTIME}(n^d)$.*

In the case that $L \notin \mathsf{BPP}$, we first need a stronger version of Lemma 13. The proof of the stronger version is identical to the proof of Lemma 13, except that we additionally use the fact that L has a worst-case to average-case reduction.

Lemma 19. *If $L \notin \mathsf{BPP}$, there are constants $\epsilon > 0$ and $a > 0$ such that for all $b > 0$, $L \notin heur_{1-n^{-a}} - \mathsf{BPTIME}(n^b + T^\epsilon)/2\log\log(T)$.*

Proof. The proof is by contradiction, and is almost identical to the proof of Lemma 13. The only change is that given a machine of size $2\log\log(T)+O(1)$ deciding L on average on inputs of length n in time T^{ϵ},we first transform it into a machine of size $2\log\log(T)+O(1)$ operating in time $T^{\epsilon}poly(n)$ and deciding L correctly on all inputs of length n, using the worst-case to average-case reduction for L. It is sufficient to pick a as in the statement of Lemma 17 to derive a contradiction (where, as before, n^k is the number of queries made by the worst-case to average-case reduction for L on inputs of length n). □

Let a be as in the statement of Lemma 17. We define a language $L_{pad'}$ which is a padded version of L such that $L_{pad'} \in heur_{1-n^{-a}} - \mathsf{BPP} - heur_{1-n^{-a}} - \mathsf{BPTIME}(n^d)$. First we require some notation. Given an integer m, there is at most one integer $n < m/2$ such that $m = n + 2^{2^z}$ for some integer z. Call m "good" if such an integer exists; denote the corresponding integer by n_m and the corresponding z by z_m. For each good m, let $H_m : \{0,1\}^m \to \{0,1\}^{n_m}$ be the function mapping each string of length m to the n_m-bit prefix of the string. Intuitively, what we are doing here is trying to use H_m to define a padded language so that for any "good" length, there are no "irrelevant" strings, unlike in the proof of Theorem 1 where any string not in the specified format was irrelevant and could be rejected. Interpreting $u \in \{0,1\}^m$ as the integer in $[1, 2^m]$ which is its position in the lexicographical order of m-bit strings, we define $shift(u) = \lfloor u(m^{4a\alpha} - m^{\alpha})/2^m \rfloor$. Let $\alpha = 6(d+1)/\epsilon$. Let $T_{mid}(n_m)$ be the median of the distribution of estimates for the running time induced by the procedure ESTIMATE in the statement of Lemma 17.

Now we are ready to define $L_{pad'}$:
$L_{pad'} = \{u \mid |u| \text{ is good}, H_{|u|}(u) \in L, |u|^{\alpha} + shift(u) \geqslant T_{mid}(n_{|u|})\}$

The definition of $L_{pad'}$ is a little complicated, so we try to give some intuition for it. The definition is similar to the definition of L_{pad} in Section 4.4, in that the length of the pad is compared to a quantity based on the running time of the optimal algorithm. The difference in the present case is that this quantity depends on the input and not just on the input length - essentially, it is a random perturbation of the corresponding quantity in Section 4.4, where the randomness is derived from the input itself. The reason we add the random perturbation is that we probabilistically compute an approximation to the quantity ourself (unlike in the proof of Theorem 1 where the advice bit told us if the pad was good), and the approximations computed on different probabilistic paths may be evenly spread about the length of the pad, resulting in the BPP-promise being destroyed. Since we add a different random perturbation to the quantity for each input and then compare to the length of the pad, for most of the inputs, the approximations of the randomized quantity will not be evenly spread about the length of the pad, and hence for most inputs the BPP-promise will not be destroyed. The probabilistic approximation of the quantity is done using the procedure ESTIMATE in the proof of Lemma 17. Once we know the pad is good, we simulate the optimal algorithm to decide whether to accept or reject the input.

Lemma 20. *If $L \notin \mathsf{BPP}$, then $L_{pad'} \in heur_{1-n^{-a}}-\mathsf{BPP}$ but $L_{pad'} \notin heur_{1-n^{-a}}-\mathsf{BPTIME}(n^d)$.*

Proof. We show that $L_{pad'} \in heur_{1-m^{-a}} - \mathsf{BPTIME}(m^{3a\alpha+3a+c+4})$ but $L_{pad'} \notin heur_{1-m^{-a}} - \mathsf{BPTIME}(m^d)$.

We show the positive inclusion first. We define a probabilistic Turing machine M accepting $L_{pad'}$ in time $m^{3a\alpha+3a+c+4}$ on average. Given u of length m, M first determines if m

is good. This check can be done in linear time. If m is not good, M rejects. Otherwise, M determines n_m and $x = H_m(u)$. If $n_m < \log(m)/3$, M runs Algorithm OPTIMAL on x and accepts if and only if OPTIMAL accepts within the allotted time. If $n_m \geqslant \log(m)/3$, M runs the procedure ESTIMATE in the proof of Lemma 17 $f(m) = m^{2a\alpha+2a+4}$ times independently on input n_m. If there is a run on which ESTIMATE does not halt within time $m^{a\alpha+a+c}$, M rejects. Otherwise, let $T_{est,i}$ be the value output by ESTIMATE on run $i, 1 \leqslant i \leqslant f(m)$. M carries out the Estimate Comparison Test - it computes $m^\alpha + shift(u)$ and checks if this value is at least the median of the values $T_{est,i}, i = 1 \ldots f(m)$. If this test fails, it rejects, otherwise it simulates Algorithm OPTIMAL for T_{est} time steps on x and accepts if and only if OPTIMAL does.

We need to prove that for at least a fraction $1 - m^{-a}$ of strings of length m, M accepts u iff $u \in L_{pad'}$. First note that if m is not good, then M rejects on all strings of length m, hence we only need to consider the case when m is good.

Let $T = T(n_m)$ and $T_{mid} = T_{mid}(n_m)$. If $n_m < \log(m)/3$, since $T < 2^{6\log(n_m)} < m^2$, there is enough time for M to simulate OPTIMAL and thus for each u of length m, M accepts u iff $u \in L_{pad'}$.

Now assume $n_m \geqslant \log(m)/3$. By Lemma 17, with probability at least $1 - 1/2^{n_m^2} \geqslant 1 - 1/m^{\log(m)/9}$, ESTIMATE halts in time Tn_m^c and outputs a value T_{est} such that $T/n_m^a < T_{est} < Tn_m^a$. We consider 2 cases - (1) $T > m^{a\alpha+a}$, and (2) $T \leqslant m^{a\alpha+a}$.

If case (1) holds, for each u of length m, $m^\alpha + shift(u) \leqslant m^{a\alpha} < T/m^a < T_{mid}$, and thus $u \notin L_{pad'}$. We show that for each u of length m, M rejects u. If there is an i, $1 \leqslant i \leqslant f(m)$ such that $T_{est,i}$ is not defined, then M rejects by definition. Conditional on $T_{est,i}$ being defined for all $1 \leqslant i \leqslant f(m)$, the Estimate Comparison Test fails with probability at least $1 - 1/m^{\Omega(\log(m))}$, and hence M rejects with probability at least 2/3, assuming m is large enough.

Now assume that case (2) holds. In this case, with probability at least $1 - m^{\Omega(\log(m))}$, $T_{est,i}$ is defined for all $1 \leqslant i \leqslant f(m)$. Given that all the $T_{est,i}$ are defined, by Lemma 17 and the Chernoff bounds of Proposition 16, with probability at least $1 - m^{-a}$ over strings of length m, the Estimate Comparison Test accepts with probability at least $1 - e^{-m}$ if $m^\alpha + shift(u) \geqslant T_{mid}$ and rejects with probability at least $1 - e^{-m}$ if $m^\alpha + shift(u) < T_{mid}$. If the Estimate Comparison Test accepts with probability at least $1 - e^{-m}$, by the assumption on T, there is enough time to simulate Algorithm OPTIMAL on input $x = H_m(u)$, and hence M accepts with probability at least 2/3 if $x \in L$ and rejects with probability at least 2/3 if $x \notin L$. If the Estimate Comparison Test rejects with probability at least $1 - e^{-m}$, M rejects with probability at least 2/3. In either case M accepts u iff $u \in L_{pad'}$. Thus we have shown that with probability at least $1 - m^{-a}$ over strings u of length m, M accepts u iff $u \in L_{pad'}$.

The proof of the negative inclusion is similar to the proof of Lemma 15. We assume $L_{pad'} \in heur_{1-m^{-a}} - \mathsf{BPTIME}(m^d)$ and show that it follows that $L \in heur_{1-n^{-a}} - \mathsf{BPTIME}(T^\epsilon + n^{2(d+1)} + n^{2(d+1)a/\alpha})/2\log\log(T)$, which is a contradiction to Lemma 19. Let M be a probabilistic Turing machine deciding $L_{pad'}$ on average in time m^{d+1} with error probability $2^{-\Omega(m)}$. We define a probabilistic advice-taking machine M' taking $< 2\log\log(T)$ bits of advice, operating within time T^ϵ and accepting L on average. Given an input x of length n, M' uses its advice to determine a good length m such that $m^\alpha > Tn^a \geqslant T_{mid}$. This requires at most $2\log(\log(T))$ bits to specify. It then generates a random string u of length m such that

$H_m(u) = x$, and outputs the majority value of n independent runs of M on u. The time taken by M' is at most $T^{3(d+1)/\alpha} n^{3(d+1)a/\alpha} + n^{3(d+1)} \leqslant T^{\epsilon} + n^{3(d+1)} + n^{6(d+1)a/\alpha}$.

We need to show that M' decides L correctly on at least a $1 - n^{-a}$ fraction of inputs of length n. For each x of length n, let p_x be the fraction of inputs u of length m such that $x = H_m(u)$ and M decides correctly whether $u \in L_{pad'}$. By the choice of the advice string, $\sum_{x\in\{0,1\}^n} p_x/2^n \geqslant 1 - m^{-a}$, which implies that for at least a $1 - 4/m^a$ fraction of strings x of length n, $p_x \geqslant 3/4$. Since $m > 2n$, this means that for at least a $1 - n^{-a}$ fraction of strings x of length n, M' accepts x with probability at least $2/3$ if $x \in L$ and rejects x with probability at least $2/3$ if $x \notin L$.

□

From Lemma 18 and Lemma 20, we obtain Theorem 2 unconditionally.

5 Open problems

The primary open problem, of course, is to prove a hierarchy theorem for completely uniform probabilistic polynomial time. But there are numerous other questions, such as obtaining an "almost-everywhere" version of our hierarchy theorem for probabilistic polynomial time with advice, or proving an analogous theorem for zero-error probabilistic classes.

It would be interesting to see whether similar ideas could be useful in understanding other promise classes, such as MA or $\mathsf{NP} \cap \mathsf{coNP}$.

On a more general note, there are perhaps further interesting connections between the theory of randomness, the theory of average-case complexity, and the theory of proof systems, which could lead to new insights and techniques.

6 Acknowledgments

We thank Dieter van Melkebeek for several enlightening discussions and Nanda Raghunathan for helpful comments on an earlier version of the paper. We also thank Madhu Sudan for telling us about his result with Goldreich and Trevisan [GST04], and Adam Klivans for a useful discussion.

References

[AS92] Noga Alon and Joel H. Spencer. *The Probabilistic Method, with an appendix on open problems by Paul Erdös*. John Wiley & Sons, 1992.

[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.

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

[BG81] Charles Bennett and John Gill. Relative to a random oracle $A, P^A \neq NP^A \neq coNP^A$ with probability 1. *SIAM Journal on Computing*, 10, 1981.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the Association for Computing Machinery*, 42:269–291, 1995.

[CNS99] Jin-Yi Cai, Ajay Nerurkar, and D. Sivakumar. Hardness and hierarchy theorems for probabilistic quasi-polynomial time. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 726–735, New York, May 1999. Association for Computing Machinery.

[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.

[FS89] Lance Fortnow and Michael Sipser. Probabilistic computation and linear time. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 148–156, 1989.

[FS97] Lance Fortnow and Michael Sipser. Retraction of "Probabilistic computation and linear time". In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, page 750, 1997.

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

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

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 134–147, 1995.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001.

[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.

[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.

[RV01] Robert Rettinger and Rutger Verbeek. Monte-carlo polynomial versus linear time – the truth-table case. *Fundamentals of Computation Theory*, 13, 2001.

[SFM78] Joel I. Seiferas, Michael J. Fischer, and Albert R. 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.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 179–190. IEEE, 1965.

[TV02] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, volume 17, 2002.

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

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