A Probabilistic-Time Hierarchy Theorem for "Slightly Non-Uniform" Algorithms

Boaz Barak*

July 14, 2002

Abstract

Unlike other complexity measures such as deterministic and nondeterministic time and space, and non-uniform size, it is not known whether probabilistic time has a strict hierarchy. For example, as far as we know it may be that **BPP** is contained in the class $\mathbf{BPtime}(n)$. In fact, it may even be that the class $\mathbf{BPtime}(n^{\log n})$ is contained in the class $\mathbf{BPtime}(n)$.

In this work we prove that a hierarchy theorem does hold for "slightly non-uniform" probabilistic machines. Namely, we prove that for every function $a : \mathbb{N} \to \mathbb{N}$ where $\log \log n \le a(n) \le \log n$, and for every constant $d \ge 1$,

$$\mathbf{BPtime}(n^d)_{/a(n)} \subsetneq \mathbf{BPP}_{/a(n)}$$

here $\mathbf{BPtime}(t(n))_{/a(n)}$ is defined to be the class of languages that are accepted by probabilistic Turing machines of running time $t(n)$ and description size $a(n)$. We actually obtain the stronger result that the class $\mathbf{BPP}_{/\log \log n}$ is *not* contained in the class $\mathbf{BPtime}(n^d)_{/\log n}$ for every constant $d \ge 1$.

We also discuss conditions under which a hierarchy theorem can be proven for *fully uniform* Turing machines. In particular we observe that such a theorem does hold if **BPP** has a complete problem.

Keywords: Probabilistic Hierarchy Theorem, Small advice, Instance Checkers

*Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. Email: `boaz@wisdom.weizmann.ac.il`

Contents

1 Introduction

Complexity theory studies the problems that can be solved using a bounded amount of computational resources. These resources include time, space, non-determinism, randomness and non-uniform advice. In general the relation between the power of *different* resources is unknown (e.g., it is not known whether or not a polynomial amount of non-determinism can be exchanged for less than exponential amount of deterministic time, as can be done by trivial exhaustive search). However, it is usually the case that one can prove that if we are given more of the same resource then we *can* solve new problems. Such theorems are called *hierarchy theorems* and exist, for example, in the case of deterministic and non-deterministic time and space, and non-uniform advice. In all these cases one can prove that for "nice" functions $f : N \to \mathbb{N}$, there are problems that can be solved using $f(n)$ amount of the resource and cannot be solved using $\frac{f(n)}{(\log f(n))^2}$ amount. In particular the following inequalities are known for every constant $d \geq 1$: $\mathbf{Dtime}(n^d) \subsetneqq \mathbf{P}$, $\mathbf{Ntime}(n^d) \subsetneqq \mathbf{NP}$, $\mathbf{Size}(n^d) \subsetneqq \mathbf{P}_{/\mathbf{poly}}$, $\mathbf{Dspace}(n^d) \subsetneqq \mathbf{PSPACE}$.

A notable exception is the case of *probabilistic time*. For probabilistic time no such theorem is known. In particular as far as we know it may be the case that $\mathbf{BPtime}(n) = \mathbf{BPP}$ and it may even be the case that $\mathbf{BPtime}(n)$ is equal to classes with probabilistic *super-polynomial* time (e.g. it may hold that $\mathbf{BPtime}(n) = \mathbf{BPtime}(n^{\log n})$). In fact, not much is known beyond the trivial observation that $\mathbf{BPtime}(n) \subsetneqq \mathbf{Dtime}(f(n)) \subseteq \mathbf{BPtime}(f(n))$ for every (time-constructible) function $f(\cdot)$ such that $f(n) = 2^{\omega(n)}$. The best unconditional bound on $\mathbf{BPtime}(n)$ is that $\mathbf{BPtime}(n) \subsetneqq \mathbf{BPtime}(f(n))$ for any (time-constructible) function $f(\cdot)$ such that $f^{(c)}(n) = 2^{\omega(n)}$ for some constant c (where $f^{(c)}$ denotes the composition of f with itself c times) [KV87].[1]

What is different with probabilistic time? The main difference seems to be that a probabilistic Turing machine that decides a $\mathbf{BPtime}(t)$ language must possess a very special property. It must hold that for any $x \in \{0,1\}^*$ either $\Pr[M(x) = 1] > 2/3$ or $\Pr[M(x) = 1] < 1/3$. It is undecidable to test whether a machine M satisfies this property and it is also unknown whether one can test whether a machine M satisfies this property for every $x \in \{0,1\}^n$ using less than 2^n steps. Because of this difference, the standard diagonalization techniques that allow us to obtain hierarchy theorems for **P**,**NP** and **PSPACE** do not work for **BPP**.[2]

Note that it is believed that **BPtime** *does* have a strict hierarchy. Indeed, Impagliazzo and Wigderson [IW97] show that if there is a problem in $\mathbf{E} = \mathbf{Dtime}(2^{O(n)})$ that requires exponential-sized circuits then for every constant d there exists a constant c such that $\mathbf{BPtime}(n^d) \subseteq \mathbf{Dtime}(n^c)$ and thus $\mathbf{BPtime}(n^d) \subsetneqq \mathbf{Dtime}(n^{c+1}) \subseteq \mathbf{BPP}$.

[1] Allender *et al* strengthen this and show that for such functions $f(\cdot)$ there exists a language $L \in \mathbf{BPtime}(f(n))$ such that every $\mathbf{BPtime}(n)$ machine will fail to decide L almost everywhere [ABHH93]. See also [CNS99, KV87] for *conditional* hierarchy theorems.

[2] The hierarchy theorem for non-uniform circuits is proven using a counting argument.

1.1 Our Results

Our main result is that a hierarchy theorem does hold for probabilistic machines that are "slightly non-uniform". That is, we define the class $\mathbf{BPtime}(t(n))_{/a(n)}$ to contain all languages that are accepted by a $t(n)$-time probabilistic Turing machine with description size $a(n)$ (or almost equivalently, all languages that are accepted by a $t(n)$-time Turing machine that gets $a(n)$ bits of advice). Our main theorem is the following:

$$\mathbf{BPP}_{/\log\log n} \not\subseteq \mathbf{BPtime}(n^d)$$

for every constant $d \geq 1$. Using a padding argument, we obtain as a corollary that for every constant $d \geq 1$ and every function $a : \mathbb{N} \to \mathbb{N}$ that satisfies

1. $\omega(\log\log n) \leq a(n) \leq \log n$
2. $a(\cdot)$ is "nice" in the sense that $a(n^{O(1)}) = \Theta(a(n))$.[3]

it holds that

$$\mathbf{BPtime}(n^d)_{/\gamma a(n)} \subsetneq \mathbf{BPtime}(n^{d+1})_{/\gamma a(n)}$$

for some constant γ.

We also explore conditions under which hierarchy theorems hold for fully uniform probabilistic time. We show (using the standard diagonalization techniques) that if $\mathbf{BPP}$ has a complete problem (under a specific notion of completeness defined below), then $\mathbf{BPtime}(n^d) \subsetneq \mathbf{BPtime}(n^{d+1})$ for every constant d. By applying the Sipser-Lautenman Theorem [Sip83, Lau83], we obtain that if $\mathbf{BPP} \supseteq \mathbf{NP}$ then $\mathbf{BPP}$ does have a complete problem (under this notion) and so in this case $\mathbf{BPtime}(n^d) \subsetneq \mathbf{BPtime}(n^{d+1})$. As a corollary we obtain that $\mathbf{BPtime}(n) \neq \mathbf{NP}$.

1.2 Techniques

Our main technique is the use of an *instance checker* [BK95] for an $\mathbf{EXP}$-complete language in order to establish a hierarchy theorem. The existence of instance checkers for such languages is implied by the existence of $\mathbf{PCP}$ proof systems for $\mathbf{EXP}$ [BFL91, AS98]. The main observation we use is that languages with an instance checker have an *optimal* algorithm in some specific sense.[4] We use an instance checker to decide whether $x \in L$ for an $\mathbf{EXP}$-complete language L, when given a set of oracles such that one of these oracles decides L. Instance checkers were used before in a similar setting by Trevisan and Vadhan [TV02].

Organization. Section 2 contains some basic notations and definitions. It also contains some basic upward scaling results that are proven using a padding

[3] Note that this bound on the rate of growth is satisfied by the function $\log n$.

[4] This is similar to the optimal algorithm for $\mathbf{NP}$, that uses the self-reducibility of $\mathbf{NP}$-complete languages [Lev73, Sch76].

argument. Section 3 contains the observation that the existence of a **BPP**-complete language implies a hierarchy theorem for **BPtime**, along with some corollaries. Section 4 contains our main result, which is a hierarchy theorem for "slightly non-uniform" probabilistic Turing machines.

2 Preliminaries

We identify a language $L \subseteq \{0,1\}^*$ with its characteristic function. That is, we say that $L(x)$ is equal to 1 if $x \in L$, and to 0 otherwise. For a probabilistic Turing machine M and an input x, we define $t_M(x)$ to be the maximum number of steps M takes when given x as input, where this maximum is over all possible coin-tosses of M. Let $t:\mathbb{N} \to \mathbb{N}$ be some function, we say that $L \in \mathbf{BPtime}(t)$ if there exists a probabilistic Turing machine M such that for any $x \in \{0,1\}^*$, $t_M(x) = O(t(|x|))$ and $\Pr[M(x) = L(x)] > \frac{2}{3}$. The class **BPP** is defined to be $\cup_{c\in\mathbb{N}}\mathbf{BPtime}(n^c)$. We say that $L \in \mathbf{Dtime}(t)$ if there exists a *deterministic* Turing machine M such that $M(x) = L(x)$ for every $x \in \{0,1\}^*$. The class **EXP** is defined to be $\cup_{c\in\mathbb{N}}\mathbf{Dtime}(2^{n^c})$.

We will sometimes identify a Turing machine M with its *description* as a string in $\{0,1\}^*$. The *description size* of M, denoted $|M|$, is the length of this string. For convenience, we assume that the description of Turing machines allows padding. That is, we assume that if M is a Turing machine with description of size k, then for every $n > k$, there exists a Turing machine M' with description size n with the same output distribution and running time as M. We will use a *universal Turing machine* $\mathcal{U}$ such that on input a Turing machine M and $x \in \{0,1\}^*$, the machine $\mathcal{U}$ runs for $poly(|M|t_M(x))$ steps, and $\mathcal{U}(M,x)$ is distributed identically to $M(x)$. For a Turing machine M, and a number m, we denote by M^m the machine M, when restricted to run for at most m steps by some time-out mechanism.

2.1 Scale-up Lemmas

Time-constructible functions. We define time-constructible functions as follows (we add the extra requirement that $n \leq f(n) \leq 2^n$ since we are only interested in functions of this form):

Definition 2.1. Let $f:\mathbb{N} \to \mathbb{N}$ be a function. We say that f is *time-constructible* if $n \leq f(n) \leq 2^n$ and there is a deterministic Turing machine M such that on input n , M runs for at most $f(n)$ steps and outputs $f(n)$.

It is a basic fact in complexity theory that inclusions for complexity classes "scale-up" (and so by the contrapositive argument separations "scale-down"). This fact, which is proved using a padding argument, will be useful to us. We will now state some specific lemmas of this form that we will use. We defer the proofs to Appendix A.

Lemma 2.2. *Let $d \geq 1$ be some constant. If* $\mathbf{BPtime}(n^d) = \mathbf{BPtime}(n^{d+1})$ *then* $\mathbf{BPtime}(n^{d+1}) = \mathbf{BPtime}(n^{(d+1)\cdot\frac{d+1}{d}})$.

By repeatedly applying Lemma 2.2, we obtain

Corollary 2.3. *Let $d \geq 1$ be some constant. Suppose that* $\mathbf{BPtime}(n^d) = \mathbf{BPtime}(n^{d+1})$. *Then,* $\mathbf{BPtime}(n^d) = \mathbf{BPP}$.

Another scaling up result is the following:

Lemma 2.4. *Let $d \geq 1$ be some constant. If* $\mathbf{BPtime}(n^d) = \mathbf{BPP}$ *then* $\mathbf{BPtime}(t(n)) = \mathbf{BPtime}(t(n)^c)$ *for every constant $c \geq 1$ and time-constructible function $t : \mathbb{N} \to \mathbb{N}$ that satisfies $t(n) \geq n^d$.*

Together, Corollary 2.3 and Lemma 2.4 imply that

Corollary 2.5. *For every $d \geq 1$, if there exists a time-constructible function $t : \mathbb{N} \to \mathbb{N}$ and a constant $c > 1$ such that $t(n) \geq n^d$ and* $\mathbf{BPtime}(t(n)) \subsetneq \mathbf{BPtime}(t(n)^c)$ *then* $\mathbf{BPtime}(n^d) \subsetneq \mathbf{BPtime}(n^{d+1})$

3 A Hierarchy Theorem Using a Complete Problem

In this section we will define a restricted type of reductions and prove that if **BPP** has a complete problem under reductions of this type then probabilistic time has a strict hierarchy. Actually, a weaker form of this section's main result holds also for the standard notion of a Karp reduction (see Remark 3.7). The observations of this section are not hard and may be known, but we still think their inclusion here is helpful.

3.1 Fixed-Polynomial Karp Completeness

We define a notion of hardness of languages that is somewhat different from the standard notion of hardness. The first difference is that we want to define **BPtime**-hardness rather than **BPP**-hardness. This means that we will say that a language L is **BPtime**-hard if for any $\mathbf{BPtime}(t)$ language L' there is a reduction between L' and L. Naturally, the complexity of the reduction itself will have to depend on the complexity of L' (i.e., on t), and a natural choice is to require that the complexity of the reduction should be at most polynomial in t. Indeed, we make this requirement and we strengthen it by requiring that the complexity of the reduction should be at most t^c where c is some fixed constant *independent* of L' and t. This may seem like a strict requirement, but we note that hard languages for other classes also satisfy a similar requirement. For example, there is a $O(t^3)$ time reduction from any $\mathbf{Ntime}(t)$ language and the language $3SAT$. The formal definition follows:

Definition 3.1. Let L be a language. We say that L is **BPtime**-*hard* if there exists a constant c such that for any time-constructible function $t : \mathbb{N} \to \mathbb{N}$ and any language $L' \in \mathbf{BPtime}(t)$ there exists a deterministic $t(|x|)^c$-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$ it holds that $x \in L' \iff f(x) \in L$.

We say that L is **BPP**-*complete* if L is **BPtime**-hard and $L \in \mathbf{BPP}$.

3.2 A Detour: Promise Problems

We will now present an example for a family of languages such that all languages in this family are **BPtime**-hard by Definition 3.1. It is conjectured that there exists a language in this family that is in **BPP** and so is **BPP**-complete. To define this family we need to recall the notion of *promise problem*.

Definition 3.2. A *promise problem* Π is a pair of sets (Π_Y, Π_N) where $\Pi_Y, \Pi_N \subseteq \{0,1\}^*$ and Π_Y , Π_N are disjoint (i.e., $\Pi_Y \cap \Pi_N = \emptyset$).

Definition 3.3. Let $t:\mathbb{N} \to \mathbb{N}$ be some function. We say that $\Pi = (\Pi_Y, \Pi_N)$ is in $\mathbf{PromiseBPtime}(t)$ if there exists a probabilistic $t(n)$-time machine M such that $x \in \Pi_Y \implies \Pr[M(x) = 1] > \frac{2}{3}$ and $x \in \Pi_N \implies \Pr[M(x) = 1] < \frac{1}{3}$. We define $\mathbf{PromiseBPP} \stackrel{def}{=} \cup_{c \in \mathbb{N}} \mathbf{PromiseBPtime}(n^c)$.

We note that unlike the case of **BPtime**, there is a hierarchy theorem for **PromiseBPtime** and in particular it is known that $\mathbf{PromiseBPtime}(n^d) \subsetneq \mathbf{PromiseBPtime}(n^{d+1})$ for every constant d.

An (important) example for a promise problem is the CIRCUIT ACCEPTANCE PROBABILITY (CAP) promise problem. The CAP promise problem is the pair $(\mathrm{CAP}_Y, \mathrm{CAP}_N)$ where CAP_Y contains all circuits C such that $\Pr_{x \in \{0,1\}^n}[C(x) = 1] > 2/3$ (where n is the number of inputs to the circuit C) and CAP_N contains all circuits C such that $\Pr_{x \in \{0,1\}^n}[C(x) = 1] < 1/3$. Clearly the problem CAP is in **PromiseBPP**.

We say that a language L is *consistent* with a promise problem $\Pi = (\Pi_Y, \Pi_N)$ if for any $x \in \{0,1\}^*$ it holds that $x \in \Pi_Y \implies x \in L$ and $x \in \Pi_N \implies x \notin L$. We have the following Lemma:

Lemma 3.4. *Let L be a language consistent with the promise problem* CAP. *Then L is* **BPtime**-*hard (in the sense of Definition 3.1).*

Proof Sketch. Every $\mathbf{BPtime}(t)$ language L' can be reduced to CAP (and therefore to L) in t^2 steps using the Cook-Levin reduction. □

We thus have the following corollary:

Corollary 3.5. *If there exists a language L such that*

1. *L is consistent with the promise problem* CAP.
2. $L \in \mathbf{BPP}$

Then there exists a **BPP**-*complete language (in the sense of Definition 3.1).*

We remark that it is reasonable to believe that there exists a language L satisfying the conditions of Corollary 3.5. In particular, Impagliazzo and Wigderson [IW97] show that if there exists a language in $\mathbf{E} = \mathbf{Dtime}(2^{O(n)})$ that requires exponential-sized circuits, then the CAP promise problem can be solved in *deterministic* polynomial-time, and so there exists a language satisfying the conditions of Corollary 3.5 in $\mathbf{P} \subseteq \mathbf{BPP}$.

It is not known whether or not the reverse direction Corollary 3.5 also holds. That is, it is not known whether the existence of a **BPP**-complete language implies that there exists a language in **BPP** that is consistent with CAP.

3.3 The Hierarchy Theorem

We now turn to proving the hierarchy theorem of this section.

Theorem 3.6. *Suppose that* **BPP** *has a complete problem (by Definition 3.1). Then there exists a constant c such that for every time constructible $t : \mathbb{N} \to \mathbb{N}$ it holds that* $\mathbf{BPtime}(t(n)) \subsetneq \mathbf{BPtime}((t(n))^c)$.

Note that by Corollary 2.5 this implies that under this assumption $\mathbf{BPtime}(n^d) \subsetneq \mathbf{BPtime}(n^{d+1})$ for every constant $d \geq 1$.

Proof .Let L be the **BPP** complete problem and let M_L be a probabilistic machine that accepts L and runs in time n^a for some constant a. We know that there exists a constant b such that for every time-constructible function $t(\cdot)$, every language in $\mathbf{BPtime}(t)$ is reducible to L using a $t(n)^b$-time deterministic reduction. For a string $i \in \{0,1\}^*$, we denote by M_i the *deterministic* Turing machine whose description is i. We denote by M_i^t the machine M_i, restricted (using some timeout mechanism) to run for at most t steps.

We define the following language $Kx \in K$ iff $M_x^{t(|x|)^b}(x) \notin L$. We claim that

1. $K \in \mathbf{BPtime}(t(n)^{O(ab)})$
2. $K \notin \mathbf{BPtime}(t(n))$.

Once we prove both items we are done. Item 1 is true because deciding K can be done by simply returning $1 - M_L(M_x^{t(|x|)^b}(x))$. This value can be computed in $t(|x|)^{O(ab)}$ steps by using the universal Turing machine $\mathcal{U}$ to compute $M_x^{t(|x|)^b}(x)$ and then by running M_L on the result.

Suppose that Item 2 was false. That is, that $K \in \mathbf{BPtime}(t(n))$. By the completeness of L, there exists an i such that for every $x \in \{0,1\}^*$, $x \in K \iff M_i^{t(|x|)^b}(x) \in L$. In particular for $x = i$ it holds that $i \in K \iff M_i^{t(|i|)^b}(i) \in L$. Yet by the definition of K this happens if and only if $i \notin K$, and so we get a contradiction. $\square$

Remark 3.7. Suppose that we followed a more standard definition of **BPtime**-hardness by allowing a *different* polynomial running time of the reduction for every language $L \in \mathbf{BPtime}$. If we look at the proof of Theorem 3.6, we can see that we will still be able to derive a meaningful result from the existence of a **BPP**-complete language under this more relaxed notion. Indeed, the same proof will yield that under this assumption it holds that $\mathbf{BPtime}(t(n)) \subsetneq \mathbf{BPtime}(f(t(n))$ for every time-constructible super-polynomial function $f : \mathbb{N} \to \mathbb{N}$ (Because the language K we defined will be in $\mathbf{BPtime}(f(t(n))$ but not in $\mathbf{BPtime}(t(n)))$. In particular, under this assumption it holds that $\mathbf{BPtime}(n) \subsetneq \mathbf{BPtime}(n^{\log n})$.

Remark 3.8. The proof of the Sipser-Lautemann Theorem [Sip83, Lau83], implies that the promise problem CAP is reducible to a language L in the polynomial hierarchy (actually in $\mathbf{\Sigma_2}$). This implies that if $\mathbf{BPP} \supseteq \mathbf{NP}$ then

BPP has a complete language (by Definition 3.1) since **BPP** $\supseteq$ **NP** implies that **BPP** $\supseteq$ **PH** and so that $L \in$ **BPP**. Combining this with Theorem 3.6, we get that **BPP** $\supseteq$ **NP** implies that **BPtime**$(n) \neq$ **BPP**. As a corollary we obtain that **BPtime**$(n) \neq$ **NP** unconditionally, since if **BPtime**$(n) =$ **NP** then by scaling up we get the equality **BPP** $=$ **NP** which would imply that both **BPP** has a complete language and **BPtime**$(n) =$ **BPP**, which is a contradiction. We note that this does not rule out the possibility that **NP** $\subsetneqq$ **BPtime**(n).

4 A Hierarchy Theorem Using Small Advice

In this section we will prove an unconditional hierarchy theorem for "slightly non-uniform" probabilistic machines. Our results will have also a (rather weak) implication on standard (uniform) probabilistic machines (see Section 4.1). We start by defining what we mean by non-uniform probabilistic machines:

Definition 4.1. Let $a : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$ be two functions and let $L \subseteq \{0,1\}^*$ be a language. We say that $L \in \mathbf{BPtime}(t(n))_{/a(n)}$ if there exists a probabilistic Turing machine M and a sequence $\{s_n\}_{n \in \mathbb{N}}$ such that

1. For any $n \in \mathbb{N}$, $|s_n| \leq a(n)$.
2. For any $x \in \{0,1\}^*$, it holds that $\Pr[M^{t(|x|)}(x, s_{|x|}) = L(x)] > \frac{2}{3}$ (recall that M^t denotes the restriction of M to t steps).

Note that the condition that for any $x \in \{0,1\}^*$ either $\Pr[M^{t(|x|)}(x,s) = 1] > \frac{2}{3}$ or $\Pr[M^{t(|x|)}(x,s) = 1] < \frac{1}{3}$ is guaranteed *only* if s is equal to the "good" advice string $s_{|x|}$. An almost equivalent formulation would be to say that $L \in \mathbf{BPtime}(t(n))_{/a(n)}$ if there exists a sequence $\{M_n\}_{n \in \mathbb{N}}$ of Turing machines, such that for every n it holds that $|M_n| \leq a(n)$ and for every $x \in \{0,1\}^n$, $\Pr[M_n^{t(n)}(x) = L(x)] > \frac{2}{3}$.

Our hierarchy theorem is:

Theorem 4.2. *For every constant $d \geq 1$, the class $\mathbf{BPP}_{/\log\log n}$ is* not *contained in the class $\mathbf{BPtime}(n^d)_{/\log n}$.*

This implies the following corollary:

Corollary 4.3. *Let $a : \mathbb{N} \to \mathbb{N}$ be a function that satisfies both $\omega(\log\log n) \leq a(n) \leq \log n$ and $a(n^{O(1)}) = \Theta(a(n))$. Then for every constant $d \geq 1$ there exists a constant γ such that*

$$\mathbf{BPtime}(n^d)_{/\gamma a(n)} \subsetneqq \mathbf{BPtime}(n^{d+1})_{/\gamma a(n)}$$

Proof Sketch. Let's look at the case $d = 1$. Suppose by contradiction that $\mathbf{BPtime}(n)_{/\gamma a(n)} = \mathbf{BPtime}(n^2)_{/\gamma a(n)}$ for all constants γ. By padding this implies that $\mathbf{BPtime}(t(n))_{/\gamma a(t(n))} = \mathbf{BPtime}(t(n)^2)_{/\gamma a(t(n))}$ for all time-constructible functions $t:\mathbb{N}\to\mathbb{N}$ and constants γ. In particular we get that

$$\begin{aligned}\mathbf{BPtime}(n)_{/a(n)} = \mathbf{BPtime}(n^2)_{/a(n)}& \\ &\supseteq \mathbf{BPtime}(n^2)_{/\gamma a(n^2)} = \mathbf{BPtime}(n^4)_{/\gamma a(n^2)}\end{aligned}$$

For some constant $\gamma < 1$ that is chosen so that $a(n) \geq \gamma a(n^2)$. Continuing in the same way we obtain that for every constant c there exists a constant γ' such that

$$\mathbf{BPtime}(n)_{/a(n)} \supseteq \mathbf{BPtime}(n^c)_{/\gamma' a(n)}$$

But since $a(n) \leq \log n$ and $a(n) = \omega(\log\log n)$ we actually obtain that

$$\mathbf{BPtime}(n)_{/\log n} \supseteq \mathbf{BPtime}(n)_{/a(n)} \supseteq \mathbf{BPtime}(n^c)_{/\gamma' a(n)} \supseteq \mathbf{BPtime}(n^c)_{/\log\log n}$$

For every constant c, or in other words $\mathbf{BPtime}(n)_{/\log n} \supseteq \mathbf{BPP}_{/\log\log n}$, contradicting Theorem 4.2. $\square$

4.1 Proof Idea for Theorem 4.2

To illustrate the proof idea, we will show a flawed approach for proving the fully uniform version of Theorem 4.2. For simplicity, we'll restrict ourselves to the case $d = 1$. That is, we will try to prove that $\mathbf{BPP} \not\subseteq \mathbf{BPtime}(n)$. We will see that there is a point in which this proof direction fails, but this point can be overcome by introducing slight non-uniformity.

Suppose, towards a contradiction, that $\mathbf{BPtime}(n) = \mathbf{BPP}$. By Lemma 2.4, this means that $\mathbf{BPtime}(t(n)) = \mathbf{BPtime}(t(n)^c))$ for every time constructible function $t(\cdot)$ and every constant $c \geq 1$. This means that for every language L, if there exists a $\mathbf{BPtime}(T(n))$ algorithm for L (for some time-constructible function $T(\cdot)$), then there exists an algorithm that solves L in $\mathbf{BPtime}(T(n)^{1/c})$ for every constant $c \geq 1$ (assuming that $T(\cdot)$ is super-polynomial). This means that every language $L \notin \mathbf{BPP}$ has no "optimal algorithm", but rather that any algorithm for L can be improved by any polynomial factor.

We see that to get a contradiction it is sufficient to provide a language $L \notin \mathbf{BPP}$ that has an *optimal algorithm* in the following sense: there exists a $\mathbf{BPtime}(T(n))$ (for some super-polynomial function $T(\cdot)$) Turing machine A that solves L and a constant c such that every other probabilistic Turing machine A' that solves L takes *at least* $T(n)^{1/c}$ steps.

How can we do that? The idea is that our optimal algorithm A will have the following general form. Algorithm A will enumerate all Turing machines of size at most $f(n)$ (where $f(n)$ is some unbounded function, e.g. $f(n) = \log n$) and will try to use each machine to solve the language L. To prove that A is indeed an optimal algorithm we will use the fact that if there indeed exists a fast Turing machine A' that solves L, then for all but finitely many inputs, the machine A' will be one of the machines enumerated by A. Therefore, we will claim that A is able to use A' to solve L and so A will be at most polynomially slower than A' (we'll choose $f(\cdot)$ such that $2^{f(n)}$ is polynomial). Of course, one detail that is missing is how exactly will our algorithm A use the machines it enumerated to solve L. Although for sufficiently large x's, one of these machines will solve L, how can we know which one does so?

In general, this does seem like a hard problem. However, recall that we are free to choose L to be any language we want. In particular, we can choose L to be a language that has an *instance checker* [BK95]. Roughly speaking, an instance checker C for a language L is an efficient procedure that when given an input x, and an oracle $O(\cdot)$, outputs a value $v \in \{0, 1, \mathsf{fail}\}$, where we are guaranteed that if $O(\cdot)$ is an oracle for L then $v = L(x)$ and otherwise with high probability $v \in \{L(x), \mathsf{fail}\}$. Clearly, given an input x and m oracles $O_1, \ldots, O_m$ such that one of these oracles solves the language L, one can use the instance checker to compute $L(x)$. Therefore if we take L to be a language that has an instance checker, we can (almost) give an *optimal algorithm* A for L. Since all **EXP**-complete languages have an instance checker [BFL91], we can just pick L to be some **EXP**-complete language. The resulting algorithm A will work as follows:

Algorithm A:

- Input: $x \in \{0,1\}^*$ (denote $n \stackrel{def}{=} |x|$).

1. For $m = 1, 2, \ldots$
2. For each probabilistic Turing machine M of size $\log n$ do:
   (a) Let $a \leftarrow C^{M^m}(x)$ (where C is the instance checker for L and M^m denotes the machine M restricted to m steps).
   (b) If $a \neq \mathsf{fail}$ output a

Note that when A halts it will with high probability halt with the correct answer. It is not hard to prove the following claim:

Claim 4.4. *If there exists a* $\mathbf{BPtime}(t)$ *algorithm* A' *for* L*, then with high probability, on input* x *algorithm* A *stops after* $poly(|x|) \cdot t(l(|x|))^2$ *steps and outputs* $L(x)$*, where* $l(n)$ *is the length of the queries the instance checker* C *makes on inputs of length* n*.*

Define the function $T : \mathbb{N} \to \mathbb{N}$ in the following way: let $T(n)$ be the minimal number k such that for every $x \in \{0,1\}^n$, $A(x)$ halts with probability at least $\frac{2}{3}$ within k steps. We will assume that $T(\cdot)$ is super-polynomial (otherwise, we can show that $\mathbf{BPP} = \mathbf{EXP}$ and then we can get a hierarchy theorem easily, e.g., by Theorem 3.6). It may seem like we're done since we have shown that L can be solved by a time $T(\cdot)$ probabilistic algorithm, but it is not in $\mathbf{BPtime}(T(n)^{1/c})$ for some universal constant $c \geq 1$ (since if it was then A would have halted sooner).[5] However we still have two problems:

[5] In this proof outline we make the simplifying assumption that $t(l(n)) = t(n)^{O(1)}$, which is the case if $l(n) = O(n)$ (e.g., if we use the instance checker of [AS98]) and $t(n)$ is a "nice" function (e.g., $t(n) = 2^{n^{\epsilon}}$). Note that the constant c is a universal constant that depends on

1. The definition of a $\mathbf{BPtime}(T(n))$ Turing machine requires the machine to stop within $T(n)$ steps *with probability* 1 and does not allow a machine that only stops within $T(n)$ steps with high probability.
2. The function $T(\cdot)$ that we defined is not necessarily time-constructible. Therefore, even if we show that $\mathbf{BPtime}(T(n)) \nsubseteq \mathbf{BPtime}(T(n)^{1/c})$ we will not be able to use Lemma 2.4 to show that this implies that $\mathbf{BPtime}(n) \nsubseteq \mathbf{BPP}$.

It turns out that if we solve the second problem we will also solve the first. This is because if $T(\cdot)$ was time-constructible then we could implement a *time-out* mechanism and so ensure that algorithm A halts within $T(n)$ steps with probability 1. Note that it would have been enough if we found a time-constructible function $T'(\cdot)$ that approximates $T(\cdot)$ from above to a polynomial power (i.e., $T'(\cdot)$ should satisfy $T(n) \leq T'(n) \leq T(n)^d$ for some constant $d \geq 1$). The problem is that it may be the case that the function $T(\cdot)$ sits in a large "gap" between the time-constructible functions. That is, it may be that the smallest time-constructible function $T'(\cdot)$ that is larger than $T(\cdot)$ is actually *much* larger than $T(\cdot)$. That is, since the function $T(\cdot)$ is arbitrary and is not necessarily time-constructible it may be the case that it cannot even be roughly approximated by a time-constructible function.

This is where we need to resort to non-uniformity. It turns out that if we add even a very small amount of advice (i.e., $\log\log n$) then any function $T(\cdot)$ where $n \leq T(n) \leq 2^n$ can be approximated by a time-constructible function.[6] This is done by simply hardwiring the number $\lceil \log\log T(n) \rceil$. This number will take at most $\log\log n$ bits to specify, and thus we see that the function $T'(n) \stackrel{def}{=} 2^{2^{\lceil \log\log T(n) \rceil}}$, that approximates $T(n)$ from above (i.e., $T(n) \leq 2^{2^{\lceil \log\log T(n) \rceil}} \leq T(n)^2$), is time-constructible with $\log\log n$ bits of advice.

This shows that there exists a language L such that $L \in \mathbf{BPtime}(T(n))_{/\log\log n}$ but $L \notin \mathbf{BPtime}(T(n)^{1/c})$, which is not so impressive, because in fact even the stronger result $\mathbf{BPtime}(T(n))_{/1} \nsubseteq \mathbf{BPtime}(T(n))$ can be easily shown to hold due to non-uniformity. However, if we look again at our algorithm A we can see that in fact it works even against slightly *non-uniform* algorithms. Indeed, Algorithm A enumerated all machines with description size $\log n$. Therefore, even if there exists a $\mathbf{BPtime}(t(n))$ algorithm A' with $\log n$ bits of advice to solve L then A will solve L within $t(n)^c$ steps. It follows that

$$\mathbf{BPtime}(T(n))_{/\log\log n} \nsubseteq \mathbf{BPtime}(T(n)^{1/c})_{/\log n}$$

By a careful padding argument,[7] similar in spirit to Lemma 2.4, we can show that

$$\mathbf{BPP}_{/\log\log n} \nsubseteq \mathbf{BPtime}(n^d)_{\log n}$$

the instance checker.

[6] We can assume that $T(n) \leq 2^n$ since we can choose the language L to be in $\mathbf{Dtime}(2^n)$.

[7] As we'll see in Section 4.2, to make things work we need to make a slight modification Algorithm A.

For every constant $d \geq 1$. We now proceed to the actual proof, that involves some issues and subtleties not mentioned in the above description.

4.2 The Actual Proof of Theorem 4.2

We define the class $\mathbf{i.o.BPP}$ to contain all languages that a probabilistic machines decides infinitely often. That is, $L \in \mathbf{i.o.BPP}$ if there exists a probabilistic polynomial-time machine such that for infinitely many n's it is the case that for all $x \in \{0,1\}^n$, $\Pr[M(x) = L(x)] > \frac{2}{3}$. We prove Theorem 4.2 by considering two cases according to whether or not $\mathbf{EXP} \subseteq \mathbf{i.o.BPP}$.

Lemma 4.5. *Suppose that* $\mathbf{EXP} \subseteq \mathbf{i.o.BPP}$. *Then for every* d, *and every* $a : \mathbb{N} \to \mathbb{N}$ *such that* $a(n) \leq n - \omega(1)$,

$$\mathbf{BPP}_{/1} \nsubseteq \mathbf{BPtime}(n^d)_{/a(n)}$$

Lemma 4.6. *Suppose that* $\mathbf{EXP} \nsubseteq \mathbf{i.o.BPP}$. *Then, for every* d,

$$\mathbf{BPP}_{/\log\log n} \nsubseteq \mathbf{BPtime}(n^d)_{/\log n}$$

Combining Lemmas 4.5 and 4.6 yields Theorem 4.2.

4.2.1 Proof of Lemma 4.5

Let $d \geq 1$ be some constant. We let L' be the following language: M is in L' (where M describes a probabilistic Turing machine) if $\Pr[M^{|M|^{d+1}}(M) = 1] < \frac{1}{2}$, where M^t denotes M restricted to running t steps. We claim that for every $\mathbf{BPtime}(n^d)$ Turing machine M and any large enough n and $s \in \{0,1\}^{a(n)}$ there exists a string $x \in \{0,1\}^n$ such that $\Pr[M^{n^d}(s,x) \neq L'(x)] > \frac{1}{3}$. Indeed, if we let x be the Turing machine that is M with the string s "hardwired" into it then we see that $L'(x) = 1 \iff \Pr[M^{n^d}(s,x) = 1] < \frac{1}{2}$.

Yet, since $L' \in \mathbf{EXP}$ and we assume that $\mathbf{EXP} \subseteq \mathbf{i.o.BPP}$, we *can* solve L' in $\mathbf{BPP}$ on an infinite set $I \subseteq \mathbb{N}$ of input sizes. We define L in the following way $L(x) = L'(x)$ if $|x| \in I$ and $L(x) = 0$ otherwise. We see that $L \in \mathbf{BPP}_{/1} \setminus \mathbf{BPtime}(n^d)_{/a(n)}$ (we use the advice bit to tell if $n \in I$). $\square$

4.2.2 Proof of Lemma 4.6

We will choose L to be a language that satisfies the following properties:

1. L is $\mathbf{EXP}$-complete.
2. L can be decided by a 2^n-time deterministic algorithm.
3. L has an *instance checker*. That is, there exists a probabilistic polynomial-time oracle machine C with output in $\{0,1,\mathsf{fail}\}$ such that for every $x \in \{0,1\}^*$ and every oracle $O(\cdot)$, $\Pr[C^O(x) \notin \{L(x), \mathsf{fail}\}] < 2^{-\Omega(|x|)}$ and $\Pr[C^L(x) = L(x)] = 1$. Furthermore, we require that the instance checker will make only queries of fixed length that is linear in $|x|$. That is, all the queries C makes are of length $c|x|$ for some constant c.

4. For convenience we will assume that L allows padding. That is, given $x \in \{0,1\}^n$ and $m > n$ it is possible to (efficiently) extend x to a string $x' \in \{0,1\}^m$ such that $x' \in L$ iff $x \in L$.

The existence of such a language follows from Arora and Safra [AS98] (they strengthen the result of Babai, Fortnow and Lund [BFL91] to obtain an instance checker with linear-sized queries, see [TV02]). We will also assume that L only contains strings whose length is a power of 2. (If L doesn't satisfy this assumption then it can be modified to do so by dropping all other strings; the padding property implies that the modified language will still satisfy the other properties.) We denote by $I \subseteq \mathbb{N}$ the set of "interesting" input lengths. That is, $I = \{2^k \mid k \in \mathbb{N}\}$. Note that the completeness of L implies that under the assumptions of the Lemma (that $\mathbf{EXP} \nsubseteq \mathbf{i.o.BPP}$) there is no probabilistic-polynomial-time algorithm that solves L on infinitely many $n \in I$. Note also that once we restrict L only to strings in I, we can assume that the constant c of the instance checker mentioned in Item 3 is a power of 2.

We define the following function $\hat{t}:\{0,1\}^* \to \mathbb{N}$. We let $\hat{t}(x)$ be the minimum number m such that there exists a probabilistic Turing machine M of size $\log m$ such that

$$\Pr[C^{M^m}(x) \neq \mathsf{fail}] > \tfrac{2}{3}$$

We define the function $t : I \to \mathbb{N}$ as follows: $t(n) = \frac{1}{n}\max_{x\in\{0,1\}^{n/c}} \hat{t}(x)$. Intuitively, we defined $t(n)$ in this way so that on one hand we will have that $L \notin \mathbf{BPtime}(t(n))$ but on the other hand we will have that (a slightly modified version of) Algorithm A described in Section 4.1 will decide L in $poly(t(n))$ time.

We have the following two claims:

Claim 4.7. *There exists a constant e such that $L \in \mathbf{BPtime}(T'(n))$ for every time-constructible function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) \geq t(cn)^e$. (Recall that cn is the size of query the instance checker for L makes on input of size n).*

Proof Sketch. If we take Algorithm A from Section 4.1 and change it so that in Step 2 it will go over all machines of size $\log m$ (instead of $\log n$) then on input $x \in \{0,1\}^n$ with high probability it will output $L(x)$ after at most $t(cn)^e$ steps for some constant e (e.g., $e = 7$). □

As a corollary we get that under our assumption, it must be the case that $t(n)$ is super-polynomial (as otherwise we would have that $L \in \mathbf{i.o.BPP}$). That is, $t(n) = n^{\omega(1)}$.

Claim 4.8. *$L \notin \mathbf{BPtime}(t(n))_{/\log t(n)}$. Furthermore, this holds for almost all input lengths in I. That is, for every large enough $n \in I$ and every Turing machine M with $\log t(n)$ bits of advice, there exists a string $x \in \{0,1\}^n$ such that* $\Pr[M(x) = L(x)] < 2/3$.

Proof Sketch. Suppose otherwise. Then it must hold that for some $n \in I$ there exists a $\log m + \log n$-sized[8] Turing machine M such that $\Pr[M^m(x) = L(x)] >$

[8] We could have used any other unbounded function instead of $\log n$.

2/3 for every $x \in \{0,1\}^n$, where $m \le t(n) = \frac{1}{n}\max_{x\in\{0,1\}^{n/c}} \hat{t}(x)$. Take $\tilde{M}$ to be the machine M with success probability amplified so that $\Pr[\tilde{M}^{\tilde{m}}(x) = L(x)] > 1 - 2^{-\Omega(n)}$, where $\tilde{m} = \frac{n}{2}m$ ($\tilde{M}$ can be described with less than $\log \tilde{m}$ bits). By the definition of the function $t(\cdot)$, there exists a string $x \in \{0,1\}^{n/c}$ such that $\hat{t}(x) > \tilde{m}$ and so for every $\log \tilde{m}$ sized Turing machine, and in particular for the machine $\tilde{M}$, the probability that $C^{\tilde{M}^{\tilde{m}}}(x) \ne \mathsf{fail}$ is at most $\frac{2}{3}$. Yet this is a contradiction since $\Pr[C^L(x) \ne \mathsf{fail}] = 1$, and by the union bound, the probability that the checker will ask a query x' and get an answer from $\tilde{M}^{\tilde{m}}$ that is different from $L(x')$ is negligible (i.e., $2^{-\Omega(n)}$). $\square$

Will we now use the language L and a padding argument to construct for every constant d, a language L' such that L' will be in $\mathbf{BPP}_{\log\log n} \setminus \mathbf{BPtime}(n^d)_{/\log n}$. As a first observation note that since $t(n) \le 2^n$, it is not hard to show that it must hold that for infinitely many $n \in I$ it holds that $t(cn) \le t(n)^{2c}$(See Claim B.1). Therefore, $t(cn)^e \le t(n)^{e'}$ for some constant e' for every $n \in J$ where J is an infinite subset of I. Let $d \ge 1$ be some constant. We consider the following language L':

$$L' = \{x\#1^l \ : \ x \in L \text{ and } |x| \in J \text{ and } l = t(|x|)^{1/d} - |x| - 1\}$$

(note that $t(|x|)^{1/d} > |x|$ since $t(\cdot)$ is super-polynomial). We claim that

1. $L' \in \mathbf{BPtime}(n^{de'})_{/\log\log n}$.
2. $L' \notin \mathbf{BPtime}(n^d)_{/\log n}$.

Once we prove both items we'll be done since we will have

$$\mathbf{BPP}_{/\log\log n} \not\subseteq \mathbf{BPtime}(n^d)_{/\log n}$$

Proof of Item 1. Suppose that we are given a string $y = x\#1^l$, where $n \stackrel{def}{=} |y| = |x| + l + 1$. We claim that we can check whether $|x| \in J$ and $n = t(|x|)^{1/d}$ using $\log\log n$ bits of advice. If we can do this then we're done since we can then run the modified Algorithm A on x for $n^{de'} = t(x)^{e'}$ steps to decide whether or not $x \in L$. Yet, we only need 1 bit of advice to check whether or not $n = t(k)^{1/d}$ for some $k \in J$. Then, we need to check if $|x| = k$.[9] Yet, this can be done by using $\lceil \log\log k \rceil$ bits of advice since k is a power of 2. Note that since $t(\cdot)$ is super-polynomial we know that $n = t(k)^{1/d} > k^2$ and so $\lceil \log\log k \rceil < \lfloor \log\log n \rfloor$.

Proof of Item 2. Suppose for contradiction that L' is in $\mathbf{BPtime}(n^d)_{/\log n}$. This means that for every $x \in \{0,1\}^*$ such that $|x| \in J$, we can decide whether or not x is a member of the language L using $(t(|x|)^{1/d})^d = t(|x|)$ steps and $\log(t(|x|)^{1/d}) = \frac{1}{d}\log t(|x|)$ bits of advice. Yet this means that the language L can be decided infinitely often in $\mathbf{BPtime}(t(n))_{/\log t(n)}$, contradicting Claim 4.8. $\square$

[9] We assume that there do not exist $k \ne k' \in J$ such that $t(k)^{1/d} = t(k')^{1/d}$. We can remove elements from J to ensure this condition while keeping it an infinite set.

Acknowledgments

I Thank Oded Goldreich and Avi Wigderson for helpful discussions. In particular, it was Avi who suggested to me the **BPP** hierarchy problem. I am also grateful to the anonymous RANDOM reviewers for their helpful comments.

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, January 1998. Preliminary version in FOCS' 92.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991. (Preliminary version in Proc. 31st FOCS.).

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the Association for Computing Machinery*, 42(1):269–291, January 1995.

[CNS99] Jin-Yi Cai, Ajay Nerurkar, and D. Sivakumar. Hardness and hierarchy theorems for probabilistic quasi-polynomial time. In ACM, editor, *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing: Atlanta, Georgia, May 1–4, 1999*, pages 726–735, New York, NY, USA, 1999. ACM Press.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[KV87] M. Karpinski and R. Verbeek. Randomness, provability, and the separation of monte carlo time and space. In Egon Börger, editor, *Computation Theory and Logic*, volume 270 of *LNCS*, pages 189–207. Springer, 1987.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, November 1983.

[Lev73] L. Levin. Universal search problems (in russian)". *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. English translation in Trakhtenbrot, B. A.: A survey of Russian approaches to Perebor (brute-force search) algorithms. Annals of the History of Computing, 6 (1984), pages. 384-400.

[Sch76] C. P. Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and R. Milner, editors, *Third International Colloquium on Automata, Languages and Programming*, pages 322–337, University of Edinburgh, 20–23 July 1976. Edinburgh University Press.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 330–335, Boston, Massachusetts, 25–27 April 1983.

[TV02] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In IEEE, editor, *Proceedings of 17th Conference on Computational Complexity, Montréal, Québec, May 21-24*. IEEE, 2002.

A Padding Lemmas

In this appendix we prove the "scaling up" lemmas of Section 2.1.

Lemma A.1 (Lemma 2.2 restated). *Let $d \geq 1$ be some constant. If* $\mathbf{BPtime}(n^d) = \mathbf{BPtime}(n^{d+1})$ *then* $\mathbf{BPtime}(n^{d+1}) = \mathbf{BPtime}(n^{(d+1)\cdot\frac{d+1}{d}})$.

Proof .Suppose that $\mathbf{BPtime}(n^{d+1}) = \mathbf{BPtime}(n^d)$ and let $L \in \mathbf{BPtime}(n^{(d+1)\cdot\frac{d+1}{d}})$. Define $L' = \{x\#1^l \mid x \in L, l = |x|^{\frac{d+1}{d}} - |x| - 1\}$. We see that $L' \in \mathbf{BPtime}(n^{d+1}) = \mathbf{BPtime}(n^d)$. However, this yields a $\mathbf{BPtime}(n^{d+1})$ algorithm for L: Given x of length n, pad x to obtain $x' = x\#1^l$ with $|x'| = m = n^{\frac{d+1}{d}}$. Then, run the $m^d = n^{d+1}$ time algorithm for to see if $x' \in L'$ $\square$

Lemma A.2 (Lemma 2.4 restated). *Let $d \geq 1$ be some constant. If* $\mathbf{BPtime}(n^d) = \mathbf{BPP}$ *then* $\mathbf{BPtime}(t(n)) = \mathbf{BPtime}(t(n)^c)$ *for every constant $c \geq 1$ and time-constructible function $t : \mathbb{N} \to \mathbb{N}$ that satisfies $t(n) \geq n^d$.*

Proof .Given $L \in \mathbf{BPtime}(t(n)^c)$, define $L' = \{x\#1^l \mid x \in L, l = t(|x|)^{1/d} - |x| - 1\}$. We see that $L' \in \mathbf{BPtime}(n^{dc}) = \mathbf{BPtime}(n^d)$ (since $\mathbf{BPtime}(n^d) = \mathbf{BPP}$). This means that $L \in \mathbf{BPtime}(t(n))$. $\square$

B Proof of Technical Claim

We prove here that for a function $t : I \to \mathbb{N}$ (where $I = \{2^k \mid k \in \mathbb{N}\}$) if $t(n) \leq 2^n$ then $t(cn) \leq t(n)^{2c}$ infinitely often.

Claim B.1. *Let $t : I \to \mathbb{N}$ be a function with $t(n) \geq 2$ such that there exist constants $c, N_0 \in I$ such that $t(cn) \geq t(n)^{2c}$ for all $n \geq N_0$. Then there exists an $n \in I$ such that $t(n) > 2^n$.*

Proof .We will prove that for all $k \geq 0$ it holds that

$$t(c^k N_0) \geq 2^{c^k 2^k} \tag{1}$$

This will imply that for $k > \log N_0$, if we let $n = c^k N_0$ then $t(n) > 2^n$.

We prove that Equation 1 holds by induction. It certainly holds for $k = 0$. For $k > 0$ we know that

$$t(c^k N_0) \geq t(c^{k-1} N_0)^{2c} \geq (2^{c^{k-1} 2^{k-1}})^{2c} = 2^{c^k 2^k}$$

$\square$