New connections between derandomization, worst-case complexity and average-case complexity

Dan Gutfreund *
Division of Engineering and Applied Sciences,
Harvard University,
Cambridge, MA 02138
danny@eecs.harvard.edu

Amnon Ta-Shma †
Computer Science Dept,
Tel-Aviv University, Israel, 69978
amnon@post.tau.ac.il

September 3, 2006

Abstract

We show that a mild derandomization assumption together with the worst-case hardness of NP implies the average-case hardness of a language in non-deterministic quasi-polynomial time. Previously such connections were only known for high classes such as EXP and PSPACE.

There has been a long line of research trying to explain our failure in proving worst-case to average-case reductions within NP [FF93, Vio03, BT03, AGGM06]. The bottom line of this research is essentially that (under plausible assumptions) black-box techniques cannot prove such results. Indeed, our proof is not black-box, as it uses a non-black-box reduction of Gutfreund, Shaltiel and Ta-Shma [GSTS05]. Furthermore, we prove using the same arguments as the above mentioned negative results, that this reduction cannot be done in a black-box way (again, under a plausible assumption). Thus our techniques show a way to bypass black-box impossibility arguments regarding worst-case to average-case reductions.

*Research supported by ONR grant N00014-04-1-0478 and NSF grant CNS-0430336. Part of this research was done while the author was at the Hebrew University.

†Research supported by the Israel Science Foundation, by the Binational Science Foundation, and by the EU Integrated Project QAP.

1 Introduction

1.1 Background

Most cryptographic primitives require at least the existence of One-Way Functions (OWFs) [IL89]. A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if

- f is computable in (a fixed) polynomial time, and,
- For every constant $c > 0$, for every polynomial time algorithm A, for all large enough input lengths,

$$\Pr_{y \in f(U_n)} [\ A(y) \in f^{-1}(y)\ ] \ < \ \frac{1}{n^c}$$

In words, f^{-1} is an NP search problem that is hard *on average* against BPP.[1] Furthermore, f can be computed by an algorithm that runs in some fixed polynomial time, while f^{-1} is hard for *all* polynomial-time algorithms, i.e., a weak algorithm B fools adversaries that are much stronger than B itself.

It is a common belief that OWFs exist, and a standard assumption in cryptography. In fact much more specific assumptions are also often made. Unfortunately, we do not know today how to prove these assumptions. In particular, it is widely recognized that proving (or disproving) the OWF conjecture is a fundamental goal (perhaps *the* fundamental goal) of cryptography.

Ofcourse, there is some rigorous work addressing these basic issues. In general, this work goes in two directions: an attempt to isolate parts of the problem (by showing for example that OWF follow from seemingly weaker or different assumptions), and an attempt to understand why current techniques fail. Let us start with the first direction. One can categorize the underlying conjectures as follows:

- (Worst-case hardness conjecture) Some function in NP is worst-case hard for some class of adversaries (usually BPP).
- (Average-case hardness conjecture) Some function in NP is average-case hard for some class of adversaries (usually BPP).
- (One-wayness conjecture) Some function in P is hard to invert by some class of adversaries (usually BPP).
- (Pseudo-randomness conjecture) Some function in P generates distributions of low entropy that are indistinguishable from uniform by some class of adversaries (usually BPP).

A natural goal of complexity (and foundations of cryptography) is to understand the relationship between these objects (given that we currently cannot prove any of them), e.g., one would like to show that the worst-case hardness conjecture implies the average-case

[1] Also, by Goldreich and Levin [GL89] it gives rise to an NP decision problem that is hard on average against BPP.

hardness conjecture. The only non-trivial relation that is known today is that the one-wayness conjecture is equivalent to the pseudo-randomness conjecture, when the class of adversaries is BPP [HILL99].

In all the above conjectures we assume that some function "fools" some class of adversaries, where the meaning of "fooling" varies (being worst-case hardness, average-case hardness, one-wayness or pseudo-randomness). The usual choice in cryptography is that the function lies in P or NP (according to the conjecture we work with), while the class of adversaries is BPP (or even BPTIME($t(n)$) for some super-polynomial $t(n)$). Thus, while the function is computable in a fixed polynomial time, the adversary may run in unbounded polynomial time, and so *has more resources than the algorithm for the function itself.* On the other hand, the standard choice in complexity theory is that computing the function takes more time than the class it fools, e.g., it may run in time n^c fooling algorithms running in time n^b for some $b << c$.

This difference is a crucial (though subtle) dividing line between cryptography and complexity, and there is no way to overestimate its significance. The canonic example for this distinction is the cryptographic pseudo-random generators (a.k.a. Blum-Micali-Yao generators) versus the pseudo-random generators that are used in complexity to derandomize probabilistic computations (a.k.a. Nisan-Wigderson generators).

Cryptography does indeed use very strong conjectures (often a function computable in P needs to fool adversaries running in sub-exponential time !) to obtain strong conclusions. But this only shows how weak is the rigorous basis which cryptography is based on. If the cryptographic conjectures are true, then the complexity variants of them (which are much weaker) should also be true. Nonetheless, it is surprising how little is known about these conjectures.

Let us recall what is known in the complexity setting. One can formulate an exponential-time analogues of the conjectures we listed above:

- (Worst-case hardness in EXP) Some function in EXP is worst-case hard for BPP.
- (Average-case hardness in EXP) Some function in EXP is average-case hard for BPP.
- (Exponential-time pseudo-random generators) For every constant $\epsilon > 0$, there exists a pseudorandom generator $G : n^\epsilon \to n$ fooling BPP, and the generator is computable in time 2^{n^ϵ} (i.e. exponential in the seed length).

Impagliazzo and Wigderson [IW98] (see also [TV02]) show (building on a long line of works such as [NW94, BFNW93, IW98, STV99]) that all three conjectures are equivalent. Their work was done in the context of understanding the power of randomness in computation, and indeed the equivalence above easily extends to include the following statement about the ability to reduce the amount of randomness used by efficient probabilistic algorithms.

- (Subexponential-time derandomization) For every probabilistic polynomial-time TM A, and a constant $\epsilon > 0$, there exists another probabilistic TM A^ϵ, that runs in time 2^{n^ϵ}, uses at most n^ϵ random coins, and behaves essentially the same as A.[2]

[2]Here we mean that A^ϵ maintains the functionality of A *on the average* in the sense of Kabanets [Kab01] (see Definition 2.3 and Hypothesis 2.1). This notion of derandomization is standard when working with hardness against uniform TM's (rather than circuits) and with generators that fool TM's (rather than circuits). See [IW98, Kab01, TV02, GSTS03] for discussions.

These beautiful and clean connections shed light on some of the fundamental questions in complexity theory regarding randomness and computational hardness of functions. Unfortunately, no such connections are known below the exponential level. Thus there is a big gap between the known results, and the questions about hardness and pseudo-randomness that we started with (on the cryptographic/polynomial level):

1. We can prove equivalence only in the complexity setting and not in the cryptographic setting, and,
2. Even in the complexity setting, we can prove equivalence only for the exponential level, and not for sub-exponential classes.

As an example for an open problem in this gap, consider the following question that lies in the complexity setting, yet in the sub-exponential regime.

Open Problem 1.1. *Does NP $\not\subseteq$ BPP imply the existence of a language in $\widetilde{NP} = NTIME(n^{O(\log n)})$ that is hard on average for BPP?*

We believe that a solution to this problem would be a breakthrough result in the field. In this paper we make a step towards this goal, by showing connections between derandomization, worst-case hardness and average-case hardness in a completely different setting of parameters to the connections obtained by [IW98, TV02].

1.2 Our main result

Stating our result very informally, we show:

Theorem 1.1. *(Informal) If*

- *NP is worst-case hard, and,*
- *Weak derandomization of BPP is possible,*

Then there exists a language in $\widetilde{NP}$ that is hard on average for BPP.

We explain what weak derandomization of BPP is, later on. We remark that there is an almost trivial proof of the theorem if we replace the weak derandomization assumption by a strong derandomization assumption, namely that BPP = P, or the assumption that pseudo-random generators exist. However, our result requires none of these assumptions, and instead uses a much weaker derandomization assumption (we explain this in Section 6).

So while we can not prove Open Problem 1.1 that the worst-case hardness of NP implies the average-case hardness of $\widetilde{\text{NP}}$, we are able to prove that the worst-case hardness of NP *together* with weak derandomization does the job.[3] We believe that this relationship between worst-case hardness, average-case hardness and derandomization, is intriguing on its own right, as it shows that highly non-trivial connections between these notions do exist below the exponential level.

[3]Our result is actually stronger than the one we state here. It gives a hard on the average language in the class NTIME($n^{\omega(1)}$) with the additional constraint that membership witnesses are of *polynomial* length, i.e. only the verification takes super-polynomial time, making it closer to NP. In particular, this class is contained in EXP. Also, standard separation techniques (such as the nondeterministic time hierarchy) can not separate this class from NP.

Admittedly, this connection is not as clean as the equivalence obtained by [IW98, TV02]. However, since establishing such equivalences below the exponential level is such an important task and seems hard to accomplish, we believe that any non-trivial step towards this goal is highly interesting. In Section 6 we explain why previous techniques do not seem to imply our result, and where the ideas we develop here (and in [GSTS05]) come into play. Another strong evidence that this work is in the right direction (or at the very least suggests a new way to look at the problem) is that the proof technique goes beyond current limitations in a fundamental way. For that, we now turn to the second line of research made on the problem, exposing limitations of current techniques.

1.3 Beyond black-box reductions

Let us first recall the definition of (randomized) black-box reductions.

Definition 1.1 (Black-box reductions)**.** *Let P, P' be two computational problems. We say that there is a black-box reduction from P to P', if there is a probabilistic polynomial-time oracle machine R, such that for every oracle A that solves P', R^A solves P. We say that the reduction is non-adaptive if R queries A non-adaptively.*

Bogdanov and Trevisan [BT03] (building on Feigenbaum and Fortnow [FF93]), show that unless every language in *co*NP can be decided by a family of polynomial size nondeterministic circuits (which implies the collapse of the polynomial-time hierarchy), there is no non-adaptive black-box reduction from deciding an NP-complete language on the worst-case to deciding an NP language on the average. Akavia et. al. [AGGM06] show that a reduction from solving NP in the worst-case to the harder task of inverting a one-way function (on the average) is even less likely, and under a certain restriction on the family of functions to invert, they also (conditionally) rule out adaptive black-box reductions. In Section 5 we present the basic argument that underlies all these results, and explain why it only rules out black-box reductions.

Let us now relax the notion of a black-box reduction. The following definition is equivalent to Definition 1.1 except that the oracle A is restricted to a class with bounded resources:

Definition 1.2 (Class-specific black-box reductions)**.** *Let P, P' be two computational problems, and $\mathcal{C}$ a class of algorithms. We say that there is a $\mathcal{C}$-black-box reduction from P to P', if there is a probabilistic polynomial-time oracle machine R such that for every oracle $A \in \mathcal{C}$ that solves P', R^A solves P. If R queries A non-adaptively we say the reduction is non-adaptive.*

One property that is common both to black-box and class-specific black-box reductions is that the reduction does not need to know the internal working of the oracle A. Conceptually, this should make these reductions easier to design. The key difference between the two types of reductions, is that in class-specific black-box reductions, the reduction (or more to the point, its analysis) needs to use the fact that A comes from some restricted class, while in black-box reductions, A can be arbitrary. Surprisingly, this makes a whole lot of difference, as we now explain.

Gutfreund, Shaltiel and Ta-Shma [GSTS05] show a non-trivial worst-case to average-case reduction within NP. Our result is based on (a derandomized variant of) their reduction. Using the terminology of class-specific black-box reductions, we can restate informally their result.

Theorem 1.2 ([GSTS05])**.** *There exists a distribution $\mathcal{D}$ samplable in time $n^{\log n}$, such that there is a* BPP-black-box *and non-adaptive reduction from solving SAT on the worst-case to solving SAT on the average with respect to $\mathcal{D}$.*

On the other hand we can also prove, using the same line of arguments as [BT03], that general black-box reductions are unlikely to achieve the parameters of the [GSTS05] reduction.

Theorem 1.3. *Suppose that there is a language $L \in NP$ and a distribution $\mathcal{D}$ samplable in time $n^{\log n}$ such that there is a* black-box *and non-adaptive reduction from solving SAT on the worst-case to solving L on the average with respect to $\mathcal{D}$. Then every language in coNP can be computed by a family of nondeterministic Boolean circuits of size $n^{polylog(n)}$.*

Theorem 1.2 is in sharp contrast to Theorem 1.3. The latter (as well as [FF93, BT03, AGGM06]) say that the requirement in black-box reductions that they succeed whenever they are given a "good" oracle (regardless of its complexity) is simply too strong, i.e. such reductions are unlikely to exist. Theorem 1.2, on the other hand, says that weakening the requirement to work only for efficient oracles, but not every oracle, is sufficient for bypassing the limitation. Thus we believe that the message from this work is that we should not be intimidated by black-box limitation results such as [FF93, BT03, AGGM06]; there are ways to overcome such arguments.

1.4 Our approach

In this section we explain the interplay between the complexities of the adversaries, the hard language and the sampler (that generates instances of the language that the adversaries cannot solve). We then explain the approach we take here and in our previous work with Ronen Shaltiel [GSTS05]. To do that we borrow terminology from game theory.

Recall that a language L is average-case hard if there exists a single samplable (in a fixed polynomial time) distribution over the instances of L, such that every probabilistic polynomial-time algorithm that attempts to decide the language, errs with high probability over instances drawn from this distribution. One way to interpret this is to consider the following game, determined by a given language L, between an algorithms player A, and a distributions player D. First D proposes a samplable distribution (described by a TM), then A answers with probabilistic polynomial-time algorithm (attempting to decide L). We say that D wins if the algorithm that A suggested, fails to decide L correctly with relatively high probability (say 1/10), over instances drawn from the distribution that D came up with. Thus the task of showing that L is average-case hard boils down to showing that D has a pure winning strategy when she plays first.[4]

Now assume SAT is worst-case hard, and let us try to prove that it is also average-case hard. Consider the following two step argument:

1. Show that D has a winning strategy when she plays second (on the game defined by SAT).

2. Apply a min-max argument and argue that one can sample from the (possibly) mixed winning strategy to derive a pure winning strategy for D when she plays first.

[4]Indeed, as we explained before, this follows the standard philosophy of cryptography in which a cryptosystem should be usable by parties that run in a fixed polynomial-time and secure against adversaries that run in arbitrary polynomial-time. For that we also need the NP relation that defines L to be computable in a fixed polynomial time.

A similar approach was used (in a non-uniform setting) to prove the hard-core set lemma of Impagliazzo [Imp95]. In [GSTS05] we were able to prove the first step. We then applied a "universal" argument to obtain a mixed winning strategy for D when she plays first. The strategy gives a single distribution that is hard for every probabilistic polynomial-time algorithm, but it is only samplable in quasi-polynomial-time. Thus we could not argue that a weak sampler running in a fixed polynomial-time, beats stronger algorithms running in arbitrary polynomial-time, which is at the heart of the average-case hardness notion of [Lev86, Imp95] (Definition 2.1).

In this paper we are able to take the next step and prove that D has a *simple* pure winning strategy when she plays first. In fact, the uniform distribution is such a strategy. However, we can only do that under an unproven derandomization assumption, and furthermore, we have to change the game along the way. I.e. instead of considering the game defined by SAT, we can only show that the uniform distribution is a winning strategy for the game defined by some language in $\widetilde{\mathrm{NP}}$.

Our proof is obtained by using a very unique property of a well known reduction of Impagliazzo and Levin [IL90], and a careful derandomization of the argument in [GSTS05]. We give a high level description of the proof in Section 3.1.

1.5 Organization

In Section 2 we give the basic definitions and notations that we use throughout the paper. In Section 3 we formally state our main result and give a top-level description of the proof. The full proof appears in Section 4. In Section 5 we discuss the issue of black-box vs. non-black-box reductions. We formally define these notions. We then describe the previous black-box limitation results and prove that these arguments extend to the reduction of [GSTS05]. Finally, we describe in detail the reduction of [GSTS05] and explain what exactly makes it non-black-box. In Section 6 we explain why previous techniques (e.g. diagonalization, or the techniques used for the exponential level) do not seem to imply our main result. We also suggest directions for further research and where we see the difficulties and hope for improvements in the future.

2 Preliminaries

We assume that the reader is familiar with standard complexity classes such as NP, BPP and RP. We need finer definitions of complexity classes where not only the running time is taken into consideration but other resources as well. $\mathrm{BPTIME}(t(n), c(n))$ is the class of languages that can be decided by randomized Turing machines that run in time $t(n)$ and use $c(n)$ random coins. We sometimes abuse notation and use $\mathrm{BPTIME}(t(n), c(n))$ also to denote the class of probabilistic procedures (that may output many bits) with these resource constraints. $\mathrm{NTIME}(t(n), w(n))$ is the class of languages that can be decided by nondeterministic Turing machines that run in time $t(n)$, and take witnesses of length $w(n)$. $\mathrm{BPTIME}(t(n))$ and $\mathrm{NTIME}(t(n))$ stand for $\mathrm{BPTIME}(t(n), t(n))$ and $\mathrm{NTIME}(t(n), t(n))$ respectively.

PPM denotes the class of probabilistic polynomial-time TM's. When dealing with probabilistic TM's, we will sometimes be interested in the confidence (over the randomness of the machine) of getting a correct answer. A *confidence level* is a function $c : \mathbb{N} \to [1/2, 1]$. Given a machine $M \in \mathrm{PPM}$ and a confidence level c, we define the function $M_c : \{0,1\}^* \to \{0, 1, *\}$ in the following way: $M_c(x) = 1$ ($M_c(x) = 0$) if M accepts (rejects) x with probability at

least $c(|x|)$ over its coins, otherwise $M_c(x) = *$. We say that M accepts (rejects) x with confidence $c(|x|)$ if $M_c(x) = 1$ ($M_c(x) = 0$), otherwise M is undecided on x with respect to confidence level c.

2.1 Average-case complexity

For a set S we denote by $x \in_R S$ that x is chosen uniformly from S. An ensemble of distributions $\mathcal{D}$ is an infinite set of distributions $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$, where $\mathcal{D}_n$ is a distribution over $\{0,1\}^n$. We denote by $\mathcal{U}$ the uniform distribution. That is, $\mathcal{U} = \{\mathcal{U}_n\}_{n\in\mathbb{N}}$, where $\mathcal{U}_n$ is the uniform distribution over $\{0,1\}^n$. Let $A(\cdot;\cdot)$ be a probabilistic TM, using $m(n)$ bits of randomness on inputs of length n. We say that A is a sampler for the distribution $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$, if for every n, the random variable $A(1^n, y)$ is distributed identically to $\mathcal{D}_n$, where the distribution is over the random string $y \in_R \{0,1\}^{m(n)}$. In particular, A always outputs strings of length n on input 1^n. If A runs in time $t(n)$ we say that $\mathcal{D}$ is samplable in time $t(n)$. If $t(n)$ is a fixed polynomial, we simply say that $\mathcal{D}$ is samplable. Often we will need to specify bounds both on the sampling time and the amount of randomness used by the sampler. To that end we say that $\mathcal{D}$ is samplable in BPTIME$(t(n), c(n))$ if the sampler for $\mathcal{D}$ is in this class of algorithms. We will sometimes abuse notations and use $\mathcal{D}$ both to denote the distribution and its sampler. A *distributional problem* is a pair $(L, \mathcal{D})$, where L is a language and $\mathcal{D}$ is an ensemble of distributions.

Definition 2.1 (Average BPP)**.** *Let $(L, \mathcal{D})$ be a distributional problem, and $s(n)$ a function from $\mathbb{N}$ to $[0,1]$. We say that $(L, \mathcal{D})$ can be* efficiently decided on the average *with success $s(n)$, and denote it by $(L, \mathcal{D}) \in Avg_{s(n)}BPP$, if there is an algorithm $A \in PPM$ such that for every large enough n, $\Pr[A(x) = L(x)] \geq s(n)$, where the probability is over an instance $x \in \{0,1\}^n$ sampled from $\mathcal{D}_n$ and the internal coin tosses of A.*

We mention that the average-case definition that we give here is from [Imp95] (there it is denoted Heur-BPP). It differs from the original definition of Levin [Lev86]. Generally speaking, all previous works about hardness amplification and worst-case to average-case reductions, and in particular those that we use here (e.g. [BT03, IL90, GSTS05]), hold under Definition 2.1.

We denote the computational problem of deciding $(L, \mathcal{D})$ on the average with success s by $(L, \mathcal{D})_s$. For a language L defined by a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ (i.e. $L = \{x : \exists y \text{ s.t. } (x,y) \in R\}$), the *search problem* associated with L, is given x, find y such $(x,y) \in R$, if such a y exist (i.e. if $x \in L$) and output 'no' otherwise. The average-case analogue of solving the search problem of $(L, \mathcal{D})$ with success s, is to solve the search problem of L with probability at least s, over an instance of L drawn from $\mathcal{D}$ (and the internal coins of the search process). We denote this computational problem by $(L, \mathcal{D})_{search,s}$.

We need to define a non-standard (and weaker) solution to search problems, by letting the searching procedure output a list of candidate witnesses rather than one, and not requiring that the algorithm recognize 'no' instances. For a langauge L defined by a binary relation R, the *list-search problem* associated with L, is given x, find a list $y_1, \ldots, y_m$ (where $m = \text{poly}(|x|)$) such that $\exists i \in [m]$ for which $(x, y_i) \in R$, if $x \in L$. Note that for $x \notin L$ we are not required to answer 'no'. We denote by $(L, \mathcal{D})_{list-search,s}$ the average-case analogue.

The reason we need this is that we are going to apply search procedures on languages in NTIME$(n^{\omega(1)}, \text{poly}(n))$. In this case an efficient procedure cannot check whether a candidate witness is a satisfying one. We therefore cannot amplify the success probability of such

procedures. On the other hand when we only require a list that contains a witness, we can apply standard amplification techniques.

2.2 Reductions

Recall the definition of (randomized) black-box reductions.

Definition 2.2 (Black-box reductions)**.** *Let P, P' be two computational problems. We say that there is a black-box reduction from P to P', if there is a probabilistic oracle machine R, such that for every oracle A that solves P', R^A solves P. We say that the reduction is non-adaptive if R queries A non-adaptively.*

In the reduction above we have not stated the running time of R. The typical choice is polynomial in the input length. However, we will also consider reductions that run in super-polynomial time. Thus we will specifically state the running time of each reduction that we use. If we do not mention the running time then the default is polynomial. Unless stated otherwise, whenever we say reduction, we mean black-box reduction.

Note that R^A is required to solve P according to the parameters of the problem. In particular if P means solving a problem with some probability of error, then the error introduced by the fact that R is probabilistic cannot exceed the error allowed by the definition of the problem. In particular if P does not allow error then typically R must be deterministic.

2.3 Indistinguishability

Following Kabanets [Kab01], we say that two probabilistic TM's are indistinguishable if no samplable distribution can output with high probability an instance on which the answers of the machines differ significantly (averaging over their randomness). Below is the formal definition.

Definition 2.3. *Let A_1 and A_2 be two probabilistic TM's outputting 0/1, such that on inputs of length n A_1 uses $m_1(n)$ random coins and A_2 uses $m_2(n)$ random coins. For $\epsilon, \delta > 0$, we say that A_1 and A_2 are (ϵ, δ)-indistinguishable, if for every samplable distribution $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ and every $n \in \mathbb{N}$,*

$$\Pr_{x\in\mathcal{D}_n}\left[\left|\Pr_{r\in_R\{0,1\}^{m_1(n)}}[A_1(x,r)=1] - \Pr_{r'\in_R\{0,1\}^{m_2(n)}}[A_2(x,r')=1]\right| > \epsilon\right] \leq \delta$$

To save on parameters, we will sometimes take ϵ to be equal to δ and then we will say that A_1, A_2 are δ-indistinguishable (meaning (δ,δ)-indistinguishable).

Using the definition above, we can formalize the derandomization hypothesis (statement (III) from the introduction):

Hypothesis 2.1. *For every probabilistic polynomial-time algorithm A, and every constant $\epsilon > 0$, there exists another probabilistic polynomial-time algorithm A^ϵ that on inputs of length n, tosses at most n^ϵ coins, and A, A^ϵ are $\frac{1}{100}$-indistinguishable.*

We mention that the statement of the hypothesis is exactly the type of derandomization obtained in [IW98] (under the assumption EXP $\neq$ BPP), with the running time of A^ϵ being 2^{n^ϵ} instead of poly(n).

3 Main result and top-down overview

We now state Theorem 1.1 in a formal way.

Theorem 3.1. *If Hypothesis 2.1 is true and SAT $\notin$ RP then there exists a language $L \in NTIME(t(n), poly(n))$, such that, $(L,\mathcal{U}) \notin Avg_{1/2+1/\log^{\alpha} n}BPP$, where $t(n) = n^{\omega(1)}$ is an arbitrary time-constructible super-polynomial function, and $\alpha > 0$ is a universal constant.*

3.1 Top-down overview

In this section we explain the intuition and give a top-down overview of the proof of Theorem 3.1. The technical details appear in Section 4. Our starting point is a theorem of Gutfreund, Shaltiel and Ta-Shma [GSTS05] that states that if NP is worst-case hard for BPP, then there exists a single distribution $\mathcal{D}_{hard}$ that is samplable in slightly super-polynomial time (for this reason we call it a "hard" distribution), and no BPP algorithm solves SAT well on average with respect to $\mathcal{D}_{hard}$. Recall that our goal is to get a simple samplable distribution $\mathcal{D}_{easy}$ (e.g. the uniform distribution) such that $(\text{SAT}, \mathcal{D}_{easy})$ is hard on the average for BPP.

Impagliazzo and Levin [IL90] showed that if there exists a *polynomial-time* samplable distribution $\mathcal{D}$ that is hard on average for some language $L \in$ NP, then there exists another language $L' \in$ NP for which the *uniform distribution* $\mathcal{U}$ is hard on average. Thus, if $(L, \mathcal{D})$ is hard on average, then so does $(L',\mathcal{U})$. We would like to use a similar reduction in our setting, and apply it on SAT and the distribution $\mathcal{D}_{hard}$.

However we immediately run into a problem because $\mathcal{D}_{hard}$ is only samplable in super-polynomial time. Let us elaborate on how the complexity of the distribution influences the reduction of [IL90]. There are several different entities to consider in this reduction: The language L, The distribution $\mathcal{D}$, The language L' we reduce to, and the reduction R itself that solves $(L, \mathcal{D})$ on average given an oracle that solves $(L',\mathcal{U})$ on average.

We can expect that both the complexity of L' as well as the complexity of the reduction R depend on $\mathcal{D}$. Indeed, using the IL reduction, the nondeterministic procedure for the language L' involves checking membership in the language L *as well as running the sampler for* $\mathcal{D}$. In our case, since $\mathcal{D}$ is samplable in super-polynomial-time, this results in L' having a super-polynomial non-deterministic complexity (see the statement of Theorem 3.1).

It seems that the same should hold for the reduction R. Indeed the reduction of [IL90] is from search problems to search problems, which means that R must handle the membership witnesses for the language L'. As we said above, this involves computing $\mathcal{D}_{hard}$ and in particular, the complexity of R is at least that of $\mathcal{D}_{hard}$. This however means that we can not deduce from the hardness on average of $(L, \mathcal{D})$ the hardness on average of $(L',\mathcal{U})$, because the hardness of $(L, \mathcal{D})$ is only against algorithms with complexity smaller than that of $\mathcal{D}$, and the reduction complexity is at least as that of $\mathcal{D}$. This should make the reduction of [IL90] useless for us.

The surprising thing, and the main observation of the paper, is that in the Impagliazzo-Levin argument, the running time of the reduction does not depend on the time complexity of $\mathcal{D}$! It only depends on the number of random coins the sampler for $\mathcal{D}$ uses: while the reduction R does look at the membership witnesses for L', the size of these witnesses is only a function of the number of random coins the sampler uses. Furthermore, during this process, R is never required to verify the witnesses and therefore does not need to run the sampler. We formalize this observation in the following lemma.

Lemma 3.1. *For every Distribution $\mathcal{D}$ samplable in BPTIME$(t(n), c(n))$, a language $L \in$ NTIME$(t_L(n), c_L(n))$ and $0 < \delta(n) < 1$, there exists $L' \in$ NTIME$(t(n)+t_L(n)+poly(n, c(n)), c(n)+c_L(n))$ such that there is a probabilistic non-adaptive reduction, R, from $(L, \mathcal{D})_{search, 1-O(\delta(n)\cdot c^2(n))}$ to $(L', \mathcal{U})_{list-search, 1-\delta(n)}$.*
Furthermore, the running time of R is poly$(n, c(n), t_L(n))$, and note that it is independent of $t(n)$.

We prove Lemma 3.1 in Section 4.1. The conclusion of the Lemma is that in order to keep the reduction efficient, we need to reduce the randomness complexity of the sampler for $\mathcal{D}_{hard}$ from [GSTS05] to a fixed polynomial, but not its time complexity. This is fortunate because while in general there is no reason to believe that we can reduce the running time of algorithms, it is widely believed that randomness can be reduced without paying much penalty in running time. To that end we use Hypothesis 2.1, and prove:

Lemma 3.2. *Assume Hypothesis 2.1 is true. Let $t(n)$ be an arbitrary super-polynomial function. There is a distribution $\mathcal{D}$ samplable in BPTIME$(O(t(n), O(n^3))$ such that,*

$$(SAT, \mathcal{D}) \in Avg_{1-1/n}BPP \Rightarrow SAT \in RP$$

We prove Lemma 3.2 in Section 4.3. Note that Hypothesis 2.1 does not seem to derandomize general distributions that are samplable in super-polynomial-time. First, Hypothesis 2.1 only derandomizes polynomial-time algorithms and only by polynomial factors. And second, Hypothesis 2.1 only applies to decision algorithms.[5] We show however that the specific distribution $\mathcal{D}_{hard}$ from [GSTS05] can be derandomized under Hypothesis 2.1. The proof is quite technical and involves getting into the details of [GSTS05]. It is given in Section 4.3.

The above two lemmas give us $1 - 1/\text{poly}(n)$ hardness on the average for the list-search version of the problem (given the worst-case hardness and the derandomization assumptions). To get $1/2 + 1/\log^{\alpha} n$ hardness on the average for the decision problem, we use generalizations of known techniques for average-case hardness amplification [Tre05]. The tricky part is doing the amplification with a reduction whose running time is $poly(n, c(n))$ and in particular, independent of $t(n)$. By using careful generalizations of [BDCGL90], Trevisan's amplification technique [Tre05] goes through. Combining it with Lemma 3.1, and the fact that we can restrict ourselves to the setting in which samplers (resp. nondeterministic computations) use polynomial amount of randomness (resp. nondeterminism), we get,

Lemma 3.3. *Let $\mathcal{D}$ be a distribution samplable in BPTIME$(t(n), poly(n))$. Let $L \in$ NTIME$(t_L(n), poly(n))$, $\gamma(n) \geq n^{-\Omega(1)}$, and $\alpha > 0$ some universal constant. There is a language $L'' \in$ NTIME$(t(n) + t_L(n) + poly(n), poly(n))$ such that there is a probabilistic non-adaptive reduction R from $(L, \mathcal{D})_{1-\gamma(n)}$ to $(L'', \mathcal{U})_{1/2+1/\log^{\alpha} n}$.*
Furthermore, the running time of R is poly$(n, t_L(n))$, and note that it is independent of $t(n)$.

The proof of Theorem 3.1 now follows by composing the last two lemmas:

[5] In general, the standard derandomization results (such as [NW94, IW97]) do not apply to probabilistic procedures that output many bits. See [DI06] (and also [KvM99, GSTS03]) for discussion about derandomizing such procedures.

Proof. (Of Theorem 3.1) Assume Hypothesis 2.1 and SAT $\notin$ RP. Let $t(n) = n^{\omega(1)}$ be an arbitrary super-polynomial function. Let $\mathcal{D} \in \text{BPTIME}(O(t(n), c(n) = O(n^3))$ be the distribution from Lemma 3.2. By that Lemma, $(\text{SAT}, \mathcal{D}) \notin Avg_{1-1/n}\text{BPP}$.

Lemma 3.3 tells us that there exists a language $L'' \in \text{NTIME}(O(t(n)), poly(n))$ such that $(\text{SAT}, \mathcal{D})_{1-1/n}$ reduces to $(L'', \mathcal{U})_{1/2+1/\log^{\alpha} n}$ and the reduction runs in time polynomial in n. The crucial fact here is that even though $t(n)$ is super-polynomial in n, the reduction is still polynomial in n. In particular we can conclude that $(L'', \mathcal{U}) \notin Avg_{1/2+1/\log^{\alpha} n}\text{BPP}$ as desired. □

4 The proof of the lemmas

In this section we give the full details of the proofs of Lemmas 3.1, 3.2, and 3.3 that appear in Section 3.1.

4.1 The proof of Lemma 3.1

In this section we prove Lemma 3.1. Parts of the proof are only a sketch as they basically repeat the arguments of [IL90] (see also [BT03]). The reduction of Impagliazzo-Levin is depicted in Figure 1. We first describe the language L' and then the reduction to it. The proof of correctness is identical to the proof of [IL90] (see also [BT03]) so we omit it.

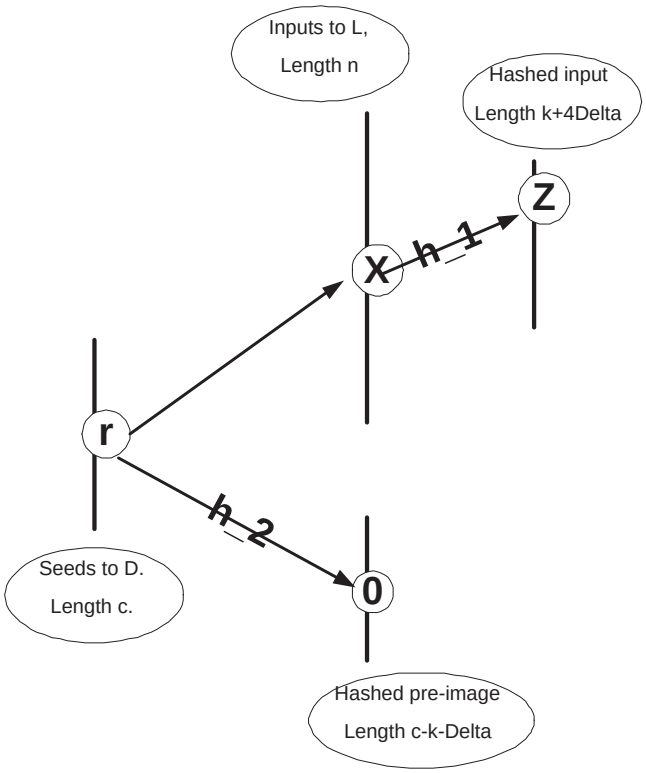

Figure 1: The Impagliazzo-Levin reduction.

4.1.1 The language L'

The language L' has instances of the form $(k, h_1, h_2, z, p) \in [c] \times \mathcal{H}_{n,k+4\Delta} \times \mathcal{H}_{c,c-k-\Delta} \times \{0,1\}^{k+4\Delta} \times \{0,1\}^{ck+(n-k)n}$, where $c = c(n)$, Δ is a small constant, $\mathcal{H}_{a,b}$ is a 2-universal family of hash-functions from $\{0,1\}^a$ to $\{0,1\}^b$ and each element in $\mathcal{H}_{a,b}$ is described by ab bits, and p is a padding string to make the input length of L' depend only on n and c but not on k. We define L' to be the set of all (k, h_1, h_2, z, p) for which there exists $r \in \{0,1\}^c$ and $w \in \{0,1\}^{c_L}$ such that

- $\mathcal{D}_n(r) \in L$ and w is a witness for that, and,
- $h_1(\mathcal{D}_n(r)) = z$ and $h_2(r) = 0$.

Clearly $L' \in \text{NTIME}(t(n) + t_L(n) + \text{poly}(n, c(n)), c(n) + c_L(n))$.

4.1.2 The reduction

Let B be an oracle that solves the list-search problem of $(L', \mathcal{U})$ with average success $1-\delta(n)$. The reduction works as follows: on input $x \in \{0,1\}^n$ to L, choose $(h_1, h_2) \in_R \mathcal{H}_{n,k+4\Delta} \times \mathcal{H}_{c,c-k-\Delta}$ and $p \in_R \{0,1\}^{ck+(n-k)n}$. Then for every $k \in [c]$ we run B on $(k, h_1, h_2, h_1(x), p)$, thus obtaining a list (of lists) of witnesses (r_i^k, w_i^k) ($1 \le k \le c$, $1 \le i \le \text{poly}(n)$, $r_i^k \in \{0,1\}^c$, $w_i^k \in \{0,1\}^{c_L}$). The reduction goes through all those witnesses and if for some $1 \le k \le c$, $1 \le i \le \text{poly}(n)$, w_i^k is a satisfying assignment for x, it outputs y_i^k and otherwise it outputs 'no'.

Notice that all the reduction has to do is choose h_1, h_2, p and compute $h_1(x)$, and it is completely oblivious to the specific distribution $\mathcal{D}$ it deals with, except for the amount of randomness it uses. In particular the reduction completely ignores r_i^k and how $\mathcal{D}$ acts on it. Thus, the reduction runs in time that is polynomial in n and $c(n)$, and independent of $t(n)$. Also, the reduction queries B in a black-box manner and non-adaptively.

4.2 Hardness amplification

The hardness amplification technique of Trevisan [Tre05] uses a search to decision reduction of Ben-David et. al. [BDCGL90]. We need a variant of this result that deals with list-search to decision reduction:

Lemma 4.1. *(list-search to decision reduction) Let $L' \in NTIME(t(n), c(n))$, and $0 < \delta(n) < 1$. There is a language $L'' \in NTIME(t(n) + poly(c(n)), c(n))$ such that there is a probabilistic non-adaptive reduction, R, from $(L', \mathcal{U})_{list-search, 1-O(\delta(n)\cdot c(n)^4)}$ to $(L'', \mathcal{U})_{1-\delta(n)}$. Furthermore, the running-time of R is poly$(n, c(n))$.*

The crucial point in the lemma is that the reduction runs in time that is polynomial in n and $c(n)$ but is independent of $t(n)$!

The proof follows the [BDCGL90] outline, and in particular uses reductions to "unique solutions". The variant that we need for the list-search scenario is:

Definition 4.1. *Let L and L' be two languages defined by the binary relations R and R' respectively. A function $\phi : \{0,1\}^n \times \{0,1\}^m \to (\{0,1\}^{n'})^\ell$ is a probabilistic reduction from L to unique solutions of L' with success probability p, if for every $x \in \{0,1\}^n$ the values $\phi(x, r) = (y^1, \ldots, y^\ell)$ are such that:*

1. *If $x \notin L$ then for every $r \in \{0,1\}^m$ and for every $1 \leq i \leq \ell$, $y^i \notin L'$. And,*
2. *If $x \in L$ then with probability at least p over $r \in_R \{0,1\}^m$, there exists an $1 \leq i \leq \ell$ such that $y^i \in L'$ and has a unique witness for that. I.e. $|\{w : R'(y^i, w)\}| = 1$.*
3. *Every witness for $y^i \in L'$ (for some $1 \leq i \leq \ell(n)$) is also a witness for $x \in L$.*

$\ell = \ell(n)$ is the list-size of the reduction.

Valiant and Vazirani [VV86] showed the following reduction to unique solutions.

Lemma 4.2. *For every language $L \in NTIME(t(n), c(n))$ there exists a language $L_{unique} \in NTIME(t(n) + poly(c(n)), c(n))$ and a probabilistic non-adaptive $poly(c(n))$-time reduction from L to unique solutions of L_{unique} with success probability $1/8$ and list-size $\ell = c(n)$.*

We can now prove Lemma 4.1.

Proof. (Of Lemma 4.1) Fix $L' \in \text{NTIME}(t(n), c(n))$. By Lemma 4.2 there exists $L'_{unique} \in \text{NTIME}(t(n) + \text{poly}(c(n)), c(n))$ and a reduction R_{unique} from L' to unique solutions of L'_{unqiue} with success probability $1/8$, list size $\ell = c(n)$ and running time $poly(c(n))$.

Let us define the language L'' that contains exactly the instances (x, r, i, j) for which there exists a witness $w \in \{0,1\}^{c(|x|)}$ such that the following two conditions are satisfied:

1. $y^i \in L'_{unique}$ and w is a witness for that, where y^i is the i'th element in $\phi(x, r) = (y^1, \ldots, y^{c(|x|)})$. And,
2. $y^i_j = 1$.

Clearly, $L'' \in \text{NTIME}(t(n) + \text{poly}(c(n)), c(n))$, and we assume here w.l.o.g. that $|y^i| \geq |x|$. We claim that indeed $(L', \mathcal{U})_{list-search, 1-O(\delta(n) \cdot c(n)^4)}$ reduces to $(L'', \mathcal{U})_{1-\delta(n)}$.

Let B be an oracle that decides $(L'', \mathcal{U})$ with average success $1 - \delta(n)$ on every input length. The reduction works as follows: on instance $x \in \{0,1\}^n$, choose $r \in \{0,1\}^m$ at random. For every $1 \leq i \leq \ell = c(n)$, for every $1 \leq j \leq c(n)$ set $w^i_j := 1$ if $B(x, r, i, j) = 1$ and 0 otherwise. Output $w^1, \ldots, w^{c(n)}$.

We now prove that the reduction works. We say that $x \in \{0,1\}^n$ is *good* if for every $i, j \in [c(n)]$,

$$\Pr_r[B(x, r, i, j) = L''(x, r, i, j)] > 1 - \frac{1}{16c(n)^2}$$

Let p_n be the fraction of good x's at length n, then,

$$(1 - p_n) \cdot \frac{1}{16c(n)^2} \cdot \frac{1}{c(n)^2} \leq \delta(n)$$

So $p_n \geq 1 - O(\delta(n) \cdot c(n)^4)$. By Lemma 4.2, for every $x \in L'$, with probability at least $1/8$ over the choice of r, there exists $i \in [c(n)]$, such that y^i has a unique witness for $y^i \in L'_{unique}$ (where $\phi(x, r) = (y^1, \ldots, y^{c(|x|)})$). Furthermore, this witness is also a witness for $x \in L'$ (by Item 3 in Definition 4.1). Whenever r defines such a y^i and $B(x, r, i, j) = L''(x, r, i, j)$ for every $i, j \in [c(n)]$, then the list $w_1, \ldots, w_{c(n)}$ contains a witness for $x \in L'$. It follows by the union bound that when $x \in L'$ is good, with probability at least $1/16$, r defines y^i with a unique witness and $B(x, r, i, j)$ answers correctly on every i, j, and so the list contains a witness for membership. We can amplify this probability by repeating this process with independent random coins at the price of somewhat increasing the length of the list. □

With that, the amplification technique of Trevisan [Tre05] goes through unchanged and gives.

Lemma 4.3. *Let $\alpha > 0$ be some universal constant. For every $L' \in NTIME(t(n), poly(n))$, and a constant $d > 0$, there exists $L'' \in NTIME(poly(t(n), n), poly(n))$ such that there is a probabilistic non-adaptive reduction, R, from $(L', \mathcal{U})_{list-search, 1-O(n^{-d} \cdot c(n)^4)}$ to $(L'', \mathcal{U})_{1/2+1/\log^{\alpha} n}$. Furthermore, the running time of R is $poly(n)$.*

We now recall that using the [IL90] reduction, we saw in Lemma 3.1 that for every distribution $\mathcal{D}$ samplable in $\text{BPTIME}(t(n), c(n))$, every $L \in \text{NTIME}(t_L(n), c_L(n))$ and $0 < \gamma(n) < 1$ there exists $L' \in \text{NTIME}(t(n) + t_L(n) + \text{poly}(n, c(n)), c(n) + c_L(n))$ such that $(L, \mathcal{D})_{1-\gamma(n)}$ reduces to $(L', \mathcal{U})_{list-search, 1-\gamma(n)/c^2(n)}$, via a reduction that runs in time $poly(n, c(n), t_L(n))$. Combining it with Lemma 4.3 we get Lemma 3.3 (where $c(n)$, $c_L(n)$, $t_L(n)$ and $1/\gamma(n)$ are all polynomial in n).

4.3 The proof of Lemma 3.2

The key lemma that we need to prove is the following:

Lemma 4.4. *If Hypothesis 2.1 is true and SAT $\notin$ RP, then for every constant $c' \geq 52/100$ and an algorithm BSAT $\in$ PPM, there is a randomized polynomial-time procedure R and a polynomial $q(\cdot)$, such that on input 1^n, the procedure R tosses $O(n^3)$ coins, and outputs at most three formulae where the length of each formula is either n or $q(n)$. Furthermore, for infinitely many input lengths n, invoking $R(1^n)$ gives with probability at least $1/10$ a set F of formulae such that there exist $\phi \in F$ with $BSAT_{c'}(\phi) \neq SAT(\phi)$.*

Once we have this lemma, we can use the arguments of [GSTS05] to obtain for every probabilistic polynomial-time algorithm, a samplable distribution that uses $O(n^3)$ random coins, and with a constant probability outputs an instance on which the algorithm errs. We can then use a "universal" argument as in [GSTS05] to obtain a single distribution that is samplable in $\text{BPTIME}(t(n), O(n^3))$ (where $t(n) = n^{\omega(1)}$) on which every probabilistic polynomial-time algorithm errs with probability at least $1/n$. This gives us the proof of Lemma 3.2. We refer the reader to [GSTS05] for the details of this part of the proof (exactly the same arguments work with the additional constraint on the randomness complexity of the samplers), and continue with the proof of Lemma 4.4. Again, as parts of the proof repeat [GSTS05] we only give here the parts in which the proofs differ. We strongly encourage the reader to read both the proof and its intuition as described in [GSTS05].

Proof. [Of Lemma 4.4] We assume that SAT $\notin$ RP and that Hypothesis 2.1 is true. Fix some constant $c' \geq 52/100$ and let $c := c' - 1/100$. Let BSAT be an arbitrary algorithm in PPM. By Hypothesis 2.1, there is an algorithm $\text{BSAT}^1 \in$ PPM that approximates BSAT well on the average (for every input length), and tosses at most n random coins on inputs of length n. we define the algorithm $\overline{\text{BSAT}}^1$ to be the amplified version of BSAT^1 as follows.

The algorithm $\overline{\text{BSAT}}^1$: When given a formula x of length n, the algorithm $\overline{\text{BSAT}}^1$ uniformly chooses n^2 independent strings $v_1, \ldots, v_{n^2}$ where each of them is of length n. For every $1 \leq i \leq n^2$, the algorithm applies $\text{BSAT}^1(x, v_i)$ and it outputs the majority vote of the answers.

Clearly, $\overline{\text{BSAT}}^1$ runs in polynomial-time and tosses n^3 coins on inputs of length n.

We now define the search algorithm SSAT^1 that uses the decision algorithm $\overline{\text{BSAT}}^1$.

The Algorithm SSAT^1: Given a formula x of length n over the variables $v_1, \ldots, v_m$, the algorithm SSAT^1 uniformly chooses a string $u \in_R \{0,1\}^{n^3}$. It then searches for a satisfying assignment for x using $\overline{\text{BSAT}}^1(\cdot, u)$ as a decision oracle (via the standard downwards self-reducibility property of SAT). SSAT^1 has three possible outputs:

- 'no', if $\overline{\text{BSAT}}^1(x,u) =$ 'no'.
- 'yes' and a Boolean assignment $\alpha = \alpha_1, \ldots, \alpha_m$, if α satisfies x.
- Three formulas of the form $x^i = x(\alpha_1, \ldots, \alpha_i, v_{i+1}, \ldots, v_m)$, $x_0^i = x(\alpha_1, \ldots, \alpha_i, 0, v_{i+2}, \ldots, v_m)$ and $x_1^i = x(\alpha_1, \ldots, \alpha_i, 1, v_{i+2}, \ldots, v_m)$ (where $\alpha_1, \ldots, \alpha_i, 0/1$ is a partial assignment), if the search fails and $\overline{\text{BSAT}}^1(\cdot, u)$ gives contradicting answers, claiming that x^i is satisfiable while x_0^i and x_1^i are not (this is the only way the search can fail).

By padding formulae before feeding them to $\overline{\text{BSAT}}^1$ we can make sure that all formulae that come up during an execution of SSAT^1 are of the length of the initial input x. We use the notation SSAT$^1(x,u)$ to describe the outcome of SSAT^1 on x when using u as random coins.

4.3.1 The first statement

For every integer n and $r \in \{0,1\}^{n^3}$ we define an NP statement $\phi_{n,r}$:

$$\exists_{x \in \{0,1\}^n} \; [\; \text{SAT}(x) = 1 \;\; \text{and} \;\; \text{SSAT}^1(x,r) \neq \text{'yes'} \;]$$

Note that indeed there is a circuit of size polynomial in n that given a formula x and an assignment α checks whether it is the case that both α satisfies x and SSAT$^1(x,r) \neq$ 'yes'. Using the Cook-Levin Theorem, we can reduce $\phi_{n,r}$ into a formula $\phi'_{n,r}$ of length $q(n)$, where q is some polynomial in n (the degree of q is determined by the running of BSAT^1 and the efficiency of the Cool-Levin reduction). The formula $\phi'_{n,r}$ is over variables x, α and z (where z is the auxiliary variables added by the reduction) with the property that $\phi'_{n,r}$ is satisfiable if and only if $\phi_{n,r}$ is satisfiable. Furthermore the Cook-Levin reduction also gives that for any triplet (x, α, z) that satisfies $\phi'_{n,r}$, x satisfies $\phi_{n,r}$ and α is a satisfying assignment for x. Let d denote the degree of the polynomial $q(n)$, and set $\epsilon := 1/d$.

4.3.2 Treating instances of length n^d

Let BSAT$^\epsilon \in$ PPM be the algorithm that approximates BSAT well on the average (for every input length), and tosses at most n^ϵ random coins on inputs of length n. Again, BSAT$^\epsilon$ exists by the assumption that Hypothesis 2.1 is true. we define the algorithm $\overline{\text{BSAT}}^\epsilon$ to be the amplified version of BSAT$^\epsilon$ as follows.

The algorithm $\overline{\text{BSAT}}^\epsilon$: When given a formula x of length n, the algorithm $\overline{\text{BSAT}}^\epsilon$ uniformly chooses $\lceil n^{2\epsilon} \rceil$ independent strings $v_1, \ldots, v_{n^{2\epsilon}}$ where each of them is of length $\lceil n^\epsilon \rceil$. For every $1 \leq i \leq \lceil n^{2\epsilon} \rceil$, the algorithm applies BSAT$^\epsilon(x, v_i)$ and it outputs the majority vote of the answers.

Clearly, $\overline{\text{BSAT}}^\epsilon$ runs in polynomial-time and tosses $\lceil n^{3\epsilon} \rceil$ coins on inputs of length n. Finally, we will need the following algorithm SSAT$^\epsilon$, which is a non-standard search algorithm that uses BSAT$^\epsilon$, and is tailored to the specific formulas $\phi'_{n,r}$ specified above.

The algorithm SSAT$^\epsilon$: We are given a string $r \in \{0,1\}^{n^3}$ and the formula $\phi'_{n,r}$ of length n^d, over the variables x, α, z as defined above. Recall that x contains n Boolean variables, α contains at most n Boolean variables and z some polynomial number of Boolean variables. The algorithm SSAT$^\epsilon$ uniformly chooses a string $u' \in_R \{0,1\}^{n^3}$ (recall that $n^3 = (n^d)^{3\epsilon}$). It then searches for a satisfying assignment for $\phi'_{n,r}$ using $\overline{\text{BSAT}}^\epsilon(\cdot, u')$ as the decision oracle. It has exactly the same three possible outputs as SSAT^1. The only difference is that SSAT$^\epsilon$ does not look for a full satisfying assignment for $\phi'_{n,r}$, but rather for an assignment to the variables in x and α that satisfies the statement $\phi_{n,r}$ (since the algorithm receives r as an input, this can be checked given assignment to x and α).

As before, by padding formulae before feeding them to $\overline{\text{BSAT}}^\epsilon$ we can make sure that all formulae that come up during an execution of SSAT$^\epsilon$ are of length n^d. We use the notation SSAT$^\epsilon(\phi'_{n,r}, u')$ to describe the outcome of SSAT$^\epsilon$ on $\phi'_{n,r}$ when using u' as random coins.

SSAT$^\epsilon$ runs in polynomial-time, and on instance $\phi'_{n,r}$ (of length n^d) tosses n^3 random coins. The reason that we needed this special type of a search algorithm that only searches for the partial assignment x, α ignoring the variables in z, is the following: $\overline{\text{BSAT}}^\epsilon$ on inputs of length n^d is an amplification of BSAT$^\epsilon$ upto confidence level $1 - 2^{-O(n)}$, and not $1 - 2^{-n^d}$ (we do not have enough coins for that). Therefore, we cannot apply Adelman-type arguments [Ade78] (as it is done in [GSTS05]) on the whole set of n^d-long inputs. However, the number of possible instances that come up in an execution of SSAT$^\epsilon$ is $2^{-O(n)}$ and therefore we can apply such arguments.

4.3.3 Finding hard instances

We are now ready to describe the procedure R. On input 1^n, R chooses at random a string $r \in_R \{0,1\}^{n^3}$. It then creates the formula $\phi'_{n,r}$, and feeds it to SSAT$^\epsilon$, with a random string $u' \in_R \{0,1\}^{n^3}$. There are a few cases to consider:

- SSAT$^\epsilon$ declares an error during the run and outputs three (or one) formulae. In this case the procedure outputs these formulae.
- SSAT$^\epsilon$ outputs 'no'. In this case the procedure outputs $\phi'_{n,r}$.
- SSAT$^\epsilon$ outputs 'yes' and a satisfying assignment (x, α) for the statement $\phi_{n,r}$. The procedure then runs SSAT$^1(x, r)$. There are three cases:
  - SSAT^1 declares an error during the run and outputs three (or one) formulae. In this case the procedure outputs these formulae.
  - SSAT^1 outputs 'no'. In this case the procedure outputs x.
  - SSAT^1 outputs 'yes'. In this case the procedure outputs 0^n.

4.3.4 Correctness

Clearly, R runs in polynomial-time and uses $O(n^3)$ random coins on input 1^n. We now prove its correctness. The following lemma is proven exactly as in [GSTS05], and we refer the reader to their proof for details. The only difference from their setting is that they consider one algorithm for both input lengths n and n^d, while here, BSAT^1 works on length

n and BSAT^ϵ on n^d. Keeping this in mind, one can follow their argument to verify the correctness of the lemma.

Lemma 4.5. *For infinitely many input lengths n, with probability at least $1 - 1/n$ the following event occurs: $R(1^n)$ outputs a set F of at most three formulae and exactly one of the following holds,*

1. *F contains only formulae of length n^d and there exists $y \in F$ such that $BSAT^\epsilon_c(y) \neq SAT(y)$.*
2. *F contains only formulae of length n and there exists $y \in F$ such that $BSAT^1_c(y) \neq SAT(y)$.*

We call an input length n useful if Lemma 4.5 holds with respect to it. We now use the fact that it is infeasible to come up with instances on which BSAT^1 and BSAT^ϵ behave very different than BSAT to conclude the proof of Lemma 4.4.

Lemma 4.6. *For infinitely many input lengths n, with probability at least $1/10$, $R(1^n)$ outputs a set F of at most three formulae (of lengths either n or n^d) such that there exists $y \in F$ such that $BSAT_{c'}(y) \neq SAT(y)$ (recall that $c' = c + 1/100$).*

which completes the proof of Lemma 4.4. □

We now turn to proving Lemma 4.6:

Proof. We say that an input length n is useful for BSAT^ϵ (resp. BSAT^1) if with probability at least $\frac{1}{2}(1-1/2n)$, $R(1^n)$ outputs a set F of at most three formulae each of length n^d (resp. n) and there exists $y \in F$ such that $\mathrm{BSAT}^\epsilon_c(y) \neq \mathrm{SAT}(y)$ (resp. $\mathrm{BSAT}^1_c(y) \neq SAT(y)$). By Lemma 4.5 every useful length is either useful for BSAT^ϵ or for BSAT^1. So for at least one of the two, there are infinitely many useful lengths. Let us assume without loss of generality that it is BSAT^ϵ.

Consider the following samplable distribution $\mathcal{G} = \{G_n\}$: on input 1^n, check whether there exists n' such that $(n')^d = n$. If so, run $R(1^{n'})$. If it outputs a set F of formulae of length n, choose a formula from F at random and output it. Otherwise output 0^n.

Whenever n' is useful for BSAT^ϵ, with probability at least $\frac{1}{2}(1 - 1/2n')$, $R(1^{n'})$ outputs a set F of at most three formulae each of length $(n')^d = n$ and there exists $y \in F$ such that

$$\mathrm{BSAT}^\epsilon_c(y) \neq \mathrm{SAT}(y) \tag{1}$$

Also, except for probability $3/100$, $R(1^n)$ outputs a set F such that for all $y \in F$,

$$\Pr[\mathrm{BSAT}(y) = 1] - \Pr[\mathrm{BSAT}^\epsilon(y) = 1]| \leq 1/100 \tag{2}$$

This is because $\mathcal{G}$ is a samplable distribution, and by Hypothesis 2.1, no samplable distribution can output with probability greater than $1/100$ an instance on which the acceptance probabilities of BSAT and BSAT^ϵ differ by more than $1/100$.

By the definition of confidence level and the fact that $c' = c + 1/100$, for every y for which both Equations 1 and 2 hold, $\mathrm{BSAT}_{c'}(y) \neq \mathrm{SAT}(y)$.

Altogether, with probability at least $1/3$, the set F that $R(1^n)$ outputs contains $y \in F$ such that $\mathrm{BSAT}_{c'}(y) \neq \mathrm{SAT}(y)$. □

5 Black-box vs. non-black-box reductions

In this section we consider the worst-case to average-case reduction from [GSTS05]. We show that under standard assumptions, their result cannot be proven via black-box and non-adaptive reductions. On the other hand [GSTS05] gives an unconditional non-black-box reduction which, as we explain below, can be preformed non-adaptively. This isolates the non-black-box property of the [GSTS05] reduction as the key ingredient in bypassing negative results along the line of [FF93, BT03, AGGM06]. With this understanding, we take a closer look at this reduction and discover that it is of a very particular form which makes it almost black-box and therefore convenient to work with.

5.1 Black-box limitations

We start by reviewing previous results about the impossibility to obtain worst-case/average-case equivalence for NP under black-box reductions. Bogdanov and Trevisan [BT03], building on Feigenbaum and Fortnow [FF93], show that unless every language in *co*NP can be decided by a family of polynomial size nondeterministic circuits (which implies the collapse of the polynomial-time hierarchy), there is no non-adaptive black-box reduction from deciding an NP-complete language in the worst-case to deciding an NP language on the average (recall Definition 2.2 in Section 2.2 of non-adaptive black-box reductions). Akavia et. al. [AGGM06] show that a reduction from solving NP in the worst-case to the harder task of inverting a one-way function on the average is even less likely, and under a certain restriction on the family of functions to invert, they also rule out (conditionally) adaptive black-box reductions.

The Feigenbaum-Fortnow protocol. Let us briefly explain the argument of Feigenbaum and Fortnow [FF93] on which the later arguments of [BT03, AGGM06] are based. The proof of [FF93] does not address general reductions. Instead, it deals with non-adaptive reductions that on input x make q non-adaptive queries. Furthermore, it assumes that for every $1 \leq i \leq q$, the i'th query is uniformly distributed. Such a reduction implies a worst-case to average-case reduction (with average-case hardness of $1 - 1/4q$ with respect to the uniform distribution). Let us call this type of reductions *special*.

The [FF93] result, generalized to arbitrary time bounds, says the following. Assume there is a language $L \in \text{NTIME}(t_L(n))$ and a special reduction R from SAT to L, such that R is computable in time $t_R(n)$. Then $\overline{\text{SAT}}$ has an Arthur-Merlin protocol in which Arthur uses non-uniform advice and runs in time $poly(t_L(t_R(n)))$. This implies that every language in coNP can be computed by a family of nondeterministic Boolean circuits of size $poly(t_L(t_R(n)))$.

The protocol is as follows. On input x, Arthur runs $k = \Theta(q^3)$ independent copies of R and thus generates qk queries (recall that R is non-adaptive). It sends all these queries to Merlin, who provides for each query an answer whether it is in L or not. Merlin also accompanies every "yes" answer with a membership witness (recall that $L \in \text{NTIME}(t_L(n))$). Arthur checks that all the certificates for the "yes" instances are correct and it then checks that all the k runs of the reduction with Merlin's answer reject x (recall that the aim is to prove that x is *not* in SAT). Finally it checks that for every $1 \leq i \leq q$, the number of "yes" answers on the i'th query in all the k executions is not significantly lower than a certain threshold. Specifically, Arthur rejects if for some $1 \leq i \leq q$, the number of "yes" answers on the i'th query (summing over all the k executions) is smaller than $pk - 2\sqrt{qk}$. Where p

is the fraction of "yes" instances of L for the relevant input length. This value is given to Arthur as a non-uniform advice.

Now, if $x \in L$ then by Chernoff and union bounds, with high probability, the number of "yes" instances on each query will be above this threshold and an honest prover will convince Arthur.

On the other hand, if $x \notin L$ then a cheating prover must give at least one wrong answer on every execution, and note that it can only cheat by saying "no" instead of "yes". In particular there exists a query i for which Merlin cheats at least k/q times. But then the number of "yes" instances on i must be at most $pk - k/q < pk - 2\sqrt{qk}$ and Arthur rejects.

Next we analyze the running time of the verifier. Since the reduction R runs in time $t_R(n)$, it follows that the length of each query that R makes is also at most $t_R(n)$. Therefore to verify membership in L (given a witness) for an instance of size $t_R(n)$, the verifier runs in time $t_L(t_R(n))$. The verifier runs $poly(q) = poly(t_R(n))$ executions of the reduction. It therefore follows that the running time of the verifier is $poly(t_L(t_R(n)))$.

Dealing with general reductions. Bogdanov and Trevisan [BT03], showed how to adapt the above argument to general non-adaptive black-box worst-case to average-case reductions that are not necessarily special. They do it in two steps. First they show how to generalize the argument of [FF93] to *smooth* reductions, in which every query is generated with probability that is at most polynomially larger than the uniform distribution. Then they reduce the general case to the smooth case. They do that by modifying the reduction such that it only asks the oracle "light" queries, i.e. queries that are not generated with too high probability. "Heavy" queries are assumed by the reduction to be 'no' instances. This makes the reduction smooth, but it also changes the oracle. However, the point is that the fraction of "heavy" queries cannot be too large. Therefore we only change the oracle on relatively few inputs. So if the original oracle solves the language well on the average, the new oracle will still solve it well on the average. Since our reduction is from solving SAT on the worst-case to solving L on the average, the new reduction with the new oracle still solves L on the worst-case, and we are in position to apply the arguments of the smooth case. We refer the reader to [BT03] for the technical details.

We want to stress out that in the proofs of [FF93, BT03] the side who is playing the oracle is Merlin. That is, the oracle has unbounded computational power. This is the reason that their arguments only rule out black-box reductions.

As in [FF93], the proof of [BT03] generalizes to arbitrary time bounds and in particular gives the following.

Theorem 5.1 (Implicit in [BT03])**.** *Suppose that there is a language $L \in NTIME(n^{O(\log n)})$ and a reduction R from solving SAT on the worst-case, to solving $(L,\mathcal{U})$ on the average with success $1 - 1/n^{O(\log n)}$. Further suppose that R is non-adaptive and black-box, and is computable in time $n^{polylog(n)}$. Then every language in coNP can be computed by a family of nondeterministic Boolean circuits of size $n^{polylog(n)}$.*

The reduction of [GSTS05] cannot be black-box. We can now prove that the reduction from [GSTS05] cannot be black-box and non-adaptive (under an unproven but plausible assumption).

Theorem 5.2. *Suppose that there is a language $L \in NP$ and a distribution $\mathcal{D}$ samplable in time $n^{\log n}$, such that there is a black-box and non-adaptive reduction from solving SAT on*

the worst-case, to solving $(L, \mathcal{D})$ *on the average with success* $1 - 1/n$*. Then every language in coNP can be computed by nondeterministic Boolean circuits of size* $n^{polylog(n)}$*.*

Proof. Using the notation from Section 2, $\mathcal{D}$ is samplable in BPTIME($n^{O(\log n)}$), and $L \in$ NTIME($poly(n)$). Then by Lemma 3.1, there is a language $L' \in$ NTIME($n^{O(\log n)}$), and a reduction R_1 from solving $(L, \mathcal{D})_{1-1/n}$ to solving $(L', \mathcal{U})_{list-search, 1-1/n^{O(\log n)}}$ (in the statement of the lemma the reduction is from $(L, \mathcal{D})_{search, 1-1/n}$, which in particular implies the decision version). This reduction is non-adaptive and black-box, and it is computable in time $n^{O(\log n)}$. By Lemma 4.1, there is a language $L'' \in$ NTIME($n^{O(\log n)}$), and a reduction R_2, from solving $(L', \mathcal{U})_{list-search, 1-1/n^{O(\log n)}}$ to solving $(L'', \mathcal{U})_{1-1/n^{O(\log n)}}$. This reduction is non-adaptive and black-box, and it is computable in time $n^{O(\log n)}$. Combining the reduction from the hypothesis of the theorem with R_1 and R_2, we obtain a reduction from solving SAT on the worst-case to solving $(L'', \mathcal{U})$ on the average with success $1 - 1/n^{O(\log n)}$. This reduction is non-adaptive and black-box, and it is computable in time $n^{polylog(n)}$ (by composing three $n^{O(\log n)}$ functions). By Theorem 5.1, this implies that every language in coNP is decidable by a family of nondeterministic circuit of quasi-polynomial size. □

5.2 A closer look at the reduction of [GSTS05]

As we said before (Section 1.4) there are two steps in [GSTS05]. First it is shown that assuming NP $\not\subseteq$ BPP, any probabilistic, polynomial-time algorithm BSAT for SAT has a hard polynomial-time samplable distribution $\mathcal{D}_{\mathrm{BSAT}}$, and then they show one quasi-polynomial time distribution $\mathcal{D}$ that is hard for all polynomial-time, probabilistic algorithms.

We now recall how the first step is achieved. Given BSAT we define the probabilistic polynomial time algorithm SSAT that tries to solve the search problem of SAT using oracle calls to BSAT (via the downwards self-reducibility property of SAT), and answers "yes" if and only if it finds a satisfying assignment. We then define the SAT formula:

$$\exists_{x \in \{0,1\}^n} \qquad [\ \mathrm{SAT}(x) = 1 \ \text{ and } \ \mathrm{SSAT}(x) \neq \text{'yes'}\ ] \tag{3}$$

The assumption NP $\not\subseteq$ BPP implies that SSAT does not solve SAT and the sentence is true. We now ask SSAT to find a satisfying assignment to it. If it fails doing so, then BSAT is wrong on one of the queries made along the way. Otherwise, the search algorithm finds a SAT sentence x on which SSAT is wrong. This, again, means that BSAT is wrong on one of the queries SSAT makes on input x. Things are somewhat more complicated because SSAT is a probabilistic algorithm and not a deterministic one, and so the above sentence is not really a SAT formula, but we avoid these technical details and refer the interested reader to [GSTS05]. In any case, we produce a small set of queries such that on at least one of the sentences in the set, BSAT is wrong, i.e., we get a polynomial-time samplable distribution on which BSAT has a non-negligible error probability.

To implement the second step, [GSTS05] define the distribution $\mathcal{D}$ that on input 1^n picks at random a machine from the set of probabilistic machines with description size at most, say, $\log n$ and runs it up to, say, $n^{\log n}$ time. We know that for any probabilistic polynomial-time algorithm BSAT there is a hard distribution $\mathcal{D}_{\mathrm{BSAT}}$, and this distribution is sampled by some polynomial-time algorithm with a fixed description size. Thus, for n large enough, we pick this machine with probability at least $1/n$ (because the description size is $\log n$) and then we output a bad input for BSAT with probability $1/n$. The sampling time for $\mathcal{D}$ is $n^{\log n}$ (or, in fact, any super-polynomial function).

We now ask: Can we interpret the [GSTS05] result as a worst-case to average-case reduction?

Indeed, given an algorithm BSAT for solving $(\mathrm{SAT}, \mathcal{D})$, the reduction is the algorithm $R^{\mathrm{BSAT}} = \mathrm{SSAT}$. The analysis shows that if BSAT indeed solves SAT well on the average with respect to $\mathcal{D}$, then it also solves it well on the average with respect to $\mathcal{D}_{\mathrm{BSAT}}$. This implies that necessarily the sentence in Eq (3) is true, i.e., SSAT solves SAT in the worst-case. In other words, the standard search to decision reduction for SAT (that uses the downwards self-reducibility property of the language) is also a worst-case to average-case reduction!

Another question is now in place: is the reduction black-box?

Looking back at Definition 2.2 we see that a reduction is called *black-box* if it has the following *two* properties:

1. (Property 1) R makes a black-box use of the adversary A (in our case BSAT). I.e., R may call A on inputs but is not allowed to look into the code of A.

2. (Property 2) The reduction is correct for any A that solves the problem (in our case SAT), putting no limitations on the nature of A. E.g. A may not even be computable.

We see that in the reduction of [GSTS05], the first condition is satisfied. R is merely the standard search-to-decision reduction for SAT which queries the decision oracle on formulas along a path of the search tree. We can replace the standard search-to-decision reduction with the one by Ben-David et. al. [BDCGL90]. The latter makes only non-adaptive queries to the decision oracle. Thus we get a non-adaptive reduction.

However, Theorem 5.2 tells us that there is no non-adaptive, black-box reduction with the parameters [GSTS05] achieve, so we must conclude that the second condition is violated. Indeed, the key point in the *analysis* of [GSTS05] works only for *efficient* oracles BSAT. This is so, because the analysis only guarantees that the distribution $\mathcal{D}$ is hard on the average for *polynomial-time* algorithms. That is, it is shown that if the search algorithm R that uses BSAT does not solve SAT in the worst-case, then we can encode the failure of R^{BSAT} as an NP statement. Here the analysis crucially uses the fact that BSAT (and therefore R^{BSAT}) is in BPP, and therefore its computation has a short description as a Boolean formula.

So let us now summarize this surprising situation: from the reduction R's point of view, it is black-box, i.e. Property 1 holds (R does not rely on the inner working of the oracle BSAT or its complexity), but for the *analysis* to work, the oracle BSAT has to be efficient, i.e. Property 2 is violated.[6] This motivates the definition of class-specific black-box reductions that we gave in the introduction and we now recall.

Definition 5.1 (Class-specific black-box reductions)**.** *Let P, P' be two computational problems, and $\mathcal{C}$ a class of algorithms. We say that there is a $\mathcal{C}$-black-box reduction from P to P', if there is a probabilistic polynomial-time oracle machine R such that for every oracle $A \in \mathcal{C}$ that solves P', R^A solves P. If R queries A non-adaptively we say the reduction is non-adaptive.*

[6] We mention that very recently, Atserias [Ats06] gave an alternative proof to [GSTS05] where he shows that even the analysis can be done almost black-box. That is, it does not need to use the description of BSAT, it only needs to know the *running time* of BSAT. In contrast, the analysis in [GSTS05] does use the description of BSAT.

Note that the definition is only meaningful when the class $\mathcal{C}$ is more powerful than R. Otherwise R can run the oracle by itself. This is indeed the case in [GSTS05] where R runs in linear time and A runs in arbitrary polynomial time.

Given this definition and the discussion about non-adaptivity above, we can restate the result of [GSTS05] as follows:

Theorem 5.3. *There exists a distribution $\mathcal{D}$ samplable in time $n^{\log n}$, such that there is a BPP-black-box and non-adaptive reduction from solving SAT on the worst-case to solving $(L, \mathcal{D})$ on the average with success $1 - 1/n$.*

Theorem 5.3 is in sharp contrast to Theorem 5.2. Theorem 5.2 (as well as [FF93, BT03, AGGM06]) say that the requirement in black-box reductions that they succeed whenever they are given a "good" oracle (regardless of its complexity) is simply too strong, i.e. such reductions are unlikely to exist. Theorem 5.3, on the other hand, says that weakening the requirement to work only for efficient oracles (i.e. to violate Property 2, but not property 1) is enough to bypass the limitation.

Class-specific black-box reductions are non-black-box yet they share some properties with black-box reductions and they seem to enjoy from all worlds:

- The reduction R makes a black-box use of the oracle A and so R is not required to "understand" the computation A does from its description.
- The correctness proof holds for every efficient A, and so the reduction has effectively the same functionality as a standard black-box reduction, and,
- It enables us to bypass black-box limitations.

In light of this we think it is not surprising that such techniques are used in the proof of Theorem 3.1. We believe such reductions are natural and provide a convenient way to bypass black-box limitations, and we expect that this methodology would find other applications.

5.3 Two remarks

We mention that there are other examples of non-black-box reductions in complexity theory that bypass black-box limitations. The fact that the polynomial-time hierarchy collapses under the assumption NP = P is proven via a non-black-box reduction from solving NP efficiently to solving Σ_2 efficiently. On the other hand if such a black-box reduction exists the polynomial-time hierarchy collapses unconditionally.[7] It is interesting, however, that in this argument both the reduction and the proof of correctness are non-black-box, because the reduction queries the NP-oracle on statements that are encodings of the computation of the oracle A. I.e. both Properties 1 and 2 are violated.

Second, we want to mention that one should not confuse black-box limitations with non-relativizing arguments. The proof of [GSTS05] (as well as the collapse of the polynomial-time hierarchy) *can* be done in a relativizing way.

[7] This was pointed out to us by Charlie Rackoff. The argument goes as follows. We will show that $\Sigma_2 \subseteq \mathrm{P}^{\mathrm{NP}}$. Given an instance x of a complete language L in Σ_2, run the reduction on it using an NP oracle. By the correctness of the reduction this decides correctly the membership of x in L. By the efficiency of the reduction, this is a procedure in P^{NP}. This argument can be easily adapted to the case of randomized reductions.

6 Discussion

In this section we discuss our approach and compare it to other possible approaches for attacking the problem. In particular we explain why standard techniques to separate complexity classes (such as diagonalization) or to prove worst-case to average-case reductions (such as the techniques that work in the exponential level) do not seem to to imply Theorem 3.1. We then discuss directions for further research, and where we see the difficulties and hope for improvements.

6.1 Can we remove the worst-case hardness assumption?

One may wonder whether the worst-case hardness of NP plays any role in our statement, or in other words whether we can prove directly that the derandomization assumption implies the average-case hardness of $\widetilde{\text{NP}}$. The first inclination is to try and prove such a result using diagonlization. Indeed with respect to deterministic polynomial-time algorithms, $\widetilde{\text{NP}}$ (and even *deterministic* quasi-polynomial time) is average-case hard by a simple diagonalization argument. However, with our current state of knowledge, BPP can contain the whole of NEXP (in the worst-case). And even if our assumption that the randomness of BPP algorithms can be reduced by polynomial factors is true, it still leaves open the possibility that BPP contains the whole of NSUBEXP. Thus we cannot use diagonalization directly to obtain the average-case hardness of $\widetilde{\text{NP}}$.

We do want to mention that a more sophisticated diagonalization argument together with a careful hierarchy collapsing argument can show that the derandomization assumption implies the *worst-case* hardness of $\widetilde{\text{NP}}$.[8] However this argument does not seem to work in the average-case setting. This is because it is not known that easiness on the average of, say NP or $\widetilde{\text{NP}}$, implies the hierarchy collapse that is needed for the argument. In fact the question of hierarchy collapses under an easiness on the average assumption is considered by Impagliazzo [Imp95] as one of the most important questions in the field of average-case complexity, and it seems to be related back to the question of worst-case/average-case connections.

Finally, we want to point out that we show hardness on the average of the class $\text{NTIME}(t(n), poly(n))$ (and not just $\widetilde{\text{NP}}$). This class is contained in EXP, and currently cannot be separated (by diagonalization or any other known technique) from NP and even RP. Thus our result is also meaningful even if we only consider hardness on the average against probabilistic polynomial-time algorithms with one-sided error.

6.2 Can we remove the derandomization assumption?

As we already mentioned in the introduction, It is known that for high classes such as EXP and PSPACE, their worst-case and average-case complexities (against small circuits or efficient probabilistic algorithms) are equivalent [BFNW93, IW97, IW98, STV99, TV02]. We want to briefly explain why the techniques that were used to prove these connections

[8] Here is a rough sketch of the proof. Assume not. In particular the class $\text{NTIME}(n^{\log n}, poly(n))$ is contained in BPP. A careful hierarchy collapsing argument (that takes the witnesses sizes into consideration) implies that $\Sigma_2(n^{\log n}, poly(n))$ (the second level analogue of $\text{NTIME}(n^{\log n}, poly(n))$) is contained in BPP. Now use the derandomization assumption to conclude that $\Sigma_2(n^{\log n}, poly(n))$ is contained in $\text{BPTIME}(poly(n), n)$. By Lautemann [Lauer] we get that $\Sigma_2(n^{\log n}, poly(n)) \subseteq \Sigma_2(poly(n), n^2)$. But this contradicts the time hierarchy for the second level of the (alternating) hierarchy.

cannot be used to remove the derandomization assumption from our statement and prove, for example, Open Problem 1.1.

All existing proofs of these worst-case/average-case connections work (explicitly or implicitly) by encoding the truth-table of an EXP complete language using a locally-decodable code. They then argue that if one can solve the encoded language on-average, then using the local decodability one can solve the EXP complete language on the worst-case. Such arguments are bound to run in exponential time (or polynomial space) because they encode the truth table of a language, and the truth table size is exponential in the input length. In other words, while this argument can start from any worst-case hard language (even, say, in NP) it only implies that there exist a language in EXP that is average-case hard. In Fact, Viola [Vio03] showed that a certain family of reductions (that includes the argument above) cannot prove worst-case to average-case connections below exponential-time. Since this is currently the only known general technique to prove worst-case to average-case reductions, we see it as a highly important task to break this exponential-time barrier, and to prove, for example, Open Problem 1.1.

6.3 Directions for future research

Our proof technique gives us hope that continuing along this line will eventually prove (in the affirmative) Open Problem 1.1. We obtain our result by using a very unique property of the reduction of [IL90]. It says that the running time of the reduction only depends on the randomness complexity of the hard distribution, but not on its running time. Can we come up with a reduction that establishes the same task (transforming arbitrary hard distributions to the uniform one), in which the running time of the reduction does not depend at all on the sampler, and only the (nondeterministic) complexity of the hard-on-average language depends on the sampler? Plugging such a reduction into our proof (now without the need to derandomize [GSTS05]) may give worst-case to average-case reductions below the exponential level.

As we have said before, we believe that proving worst-case/average-case equivalence within NP requires completely different ideas. This is because [GSTS05] only gives a quasi-polynomial time sampler, and we don't expect a reduction, transforming arbitrary hard distributions to the uniform one, in which the complexity of both the reduction and the hard-on-average language are independent of the sampler.

Finally, we want to mention another question that is related to our work. It is well known that pseudorandom generators do imply average-case hardness (almost by definition). Note however, that in Theorem 3.1, we only use a derandomization assumption, which is seemingly weaker than the existence of pseudorandom generators (e.g. it is possible that one can show how to derandomize probabilistic algorithms without constructing a pseudorandom generator along the way). This brings us to the question of the connections between derandomization and pseudorandom generators. Does *efficient* deranodmization imply *efficient* pseudorandom generators? We know that the answer is yes when we remove the efficiency requirement [IW98].

7 Acknowledgements

We thank Ronen Shaltiel and Salil Vadhan for many helpful discussions. Andrej Bogdanov for pointing out to us that Theorem 5.2 follows directly from a generalization of [BT03] (i.e. Theorem 5.1). Previously we had a more complicated and weaker statement. Charlie

Rackoff for suggesting the relevance of the argument that the polynomial-time hierarchy collapses under P = NP. And Alex Healy for commenting on the manuscript.

References

[Ade78] L. Adelman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.

[AGGM06] A. Akavia, O. Goldreich, S. Goldwasser, and D. Moshkovitz. On basing one-way functions on np-hardness. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (to appear)*, 2006.

[Ats06] A. Atserias. Non-uniform hardness for np via black-box adversaries. In *Proceedings of the 21th Annual IEEE Conference on Computational Complexity (to appear)*, 2006.

[BDCGL90] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 379–386, 1990.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulation unless Exptime has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BT03] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 308–317, 2003.

[DI06] B. Dubrov and Y. Ishai. On the randomness complexity of efficient sampling. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (to appear)*, 2006.

[FF93] J. Feigenbaum and L. Fortnow. Random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[GSTS03] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. Uniform hardness vs. randoness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12:85–130, 2003.

[GSTS05] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. if NP languages are hard in the worst-case then it is easy to find their hard instances. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 243–257, 2005.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 230–235, 1989.

[IL90] R. Impagliazzo and L. Levin. No better ways of finding hard NP-problems than picking uniformly at random. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 812–821, 1990.

[Imp95] R. 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] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63 (2):236–252, 2001.

[KvM99] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 659–667, 1999.

[Lauer] C. Lautemann. BPP and the polynomial hierarchy. *Inf. Process. Lett.*, 17(4):215–217, 1983, November.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15 (1):285–286, 1986.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[Tre05] L. Trevisan. On uniform amplification of hardness in NP. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, 2005.

[TV02] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 129–138, 2002.

[Vio03] E. Viola. Hardness vs. randomness within alternating time. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 53–62, 2003.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(1):85–93, 1986.