Distinguishing SAT from Polynomial-Size Circuits, through Black-Box Queries

Albert Atserias[*]
Universitat Politècnica de Catalunya
Barcelona, Spain
atserias@lsi.upc.edu

Abstract

We may believe SAT does not have small Boolean circuits. But is it possible that some language with small circuits looks indistiguishable from SAT to every polynomial-time bounded adversary? We rule out this possibility. More precisely, assuming SAT does not have small circuits, we show that for every language A with small circuits, there exists a probabilistic polynomial-time algorithm that makes black-box queries to A, and produces, for a given input length, a Boolean formula on which A differs from SAT. A key step for obtaining this result is a new proof of the main result by Gutfreund, Shaltiel, and Ta-Shma reducing average-case hardness to worst-case hardness via uniform adversaries that know the algorithm they fool. *The new adversary we construct has the feature of being black-box on the algorithm it fools, so it makes sense in the non-uniform setting as well. Our proof makes use of a refined analysis of the learning algorithm of Bshouty et al..*

1 Introduction

The motivation and starting point for this article is the recent result of Gutfreund, Shaltiel and Ta-Shma [6] showing that if SAT is worst-case hard for probabilistic polynomial-time algorithms, then, for every probabilistic polynomial-time algorithm trying to solve SAT, there exists a polynomially samplable distribution that is hard for it. In the notation invented by Kabanets:

$$\mathbf{NP} \not\subseteq \mathbf{BPP} \text{ implies } \mathbf{NP} \not\subseteq \textbf{pseudo-}\mathbf{BPP}. \quad (1)$$

Here, $\mathbf{BPP}$ denotes the class of all languages accepted by probabilistic polynomial-time algorithms with bounded error, and the notation on the right **pseudo-BPP** denotes the class of all languages that look indistinguishable from some language in $\mathbf{BPP}$ from the point of view of any probabilistic polynomial-time adversary. By this we mean that the adversary is not able to produce, with non-negligible probability, any instance where the languages differ on a given input-length.

Statement (1) consitutes a sort of worst-case to average-case reduction for $\mathbf{NP}$. To see why, read it in its contrapositive form: if SAT is indistinguishable from some $\mathbf{BPP}$-language for any efficiently samplable distribution, then it is actually a $\mathbf{BPP}$-language. This was the first such reduction of any kind for $\mathbf{NP}$. The authors in [6] stress, however, that the average-case hardness they obtain for SAT is not enough for the existence of one-way functions and cryptography. Indeed, a necessary condition for cryptography is the existence of a *single* samplable distribution that is hard for all efficient algorithms *simultaneously*. What (1) shows, instead, is the existence of a samplable distribution that strongly depends on the efficient algorithm A that it is aimed to fool. As a matter of fact, the proof shows that the algorithm computing the distribution both simulates A and is strongly non-black-box on A as it explicitely needs its code. We refer the reader to the introduction of [6] for a thorough discussion on the different versions of average-case hardness, and of worst-case to average-case reductions, and their relevance to cryptography.

The first contribution of this paper is a new proof of (1) that has the good feature of producing an adversary that is essentially black-box on A, the $\mathbf{BPP}$-algorithm that supposedly solves SAT and that it aims to fool. What this means is that the adversary gets the information it needs through *oracle calls* to A. In particular, it does not know the code of A and cannot base the production of a hard instance on it. This partially solves a question left open in [6] where it is asked whether reductions that are non-black-box both in the hard language (here SAT) and in the algorithm we want to fool are necessary to prove (1). As pointed out before, our reduction is essentially black-box in the algorithm, but not in the hard language as we use the downward self-reducibility of SAT in a strong way. The technical details

[*] Supported in part by CICYT TIN2004-04343 and by the European Commission through the RTN COMBSTRU HPRN-CT2002-00278.

about the exact sense in which our sampler is black-box on the algorithm are discussed later in this introduction.

The main point of our new proof is, however, that the technique extends easily to Boolean circuits. Thus, we show that if SAT does not have polynomial-size circuits, then for every language A with polynomial-size circuits there exists a probabilistic polynomial-time algorithm that makes black-box queries to A and produces, with non-negligible probability and for a given input-length, a formula on which A differs from SAT. In the notation we introduce in this paper:

$$\mathbf{NP} \not\subseteq \mathbf{P/poly} \text{ implies } \mathbf{NP} \not\subseteq \textbf{bb-pseudo-}\mathbf{P/poly}.$$

The **bb-pseudo-** notation is a natural analogue of the **pseudo-** notation when we want to fool non-uniform algorithms with a uniform adversary (see Section 2 for details).

Overview of the proof Let us first discuss a simplified version of the proof in [6] to see where it fails to be black-box. Assume $\mathbf{NP} \not\subseteq \mathbf{P}$ and let A be any polynomial-time algorithm that attempts to solve SAT. Let us consider the following statement $S(A,n)$, parameterized by the formula-size n and the algorithm A:

> *There exists a Boolean formula F of size n such that, either $A(F) = 0$ and F is satisfiable, or $A(F) = 1$ and $A(F|_{x_1=0}) = A(F|_{x_1=1}) = 0$, or $A(F) = 1$ and F has no free variables and evaluates to* 0.

Here, $F|_{x_1=a}$ stands for the result of replacing the first free variable x_1 of F by a. By our assumption that $\mathbf{NP} \not\subseteq \mathbf{P}$, this statement is true for infinitely many n as it asserts that A is incorrect for SAT on formulas of size n. Moreover, since A is a polynomial-time algorithm, $S(A,n)$ is an $\mathbf{NP}$ statement, which means that we can construct a polynomial-size Boolean formula $G(A,n)$ that is satisfiable if and only if $S(A,n)$ is true. Now, let us run A on $G = G(A,n)$ for an n on which the statement is true. If $A(G) = 0$, then A is wrong on G. If $A(G) = 1$, then we may run the downward self-reducibility of SAT until either we find a satisfying assignment for G, or we find three formulas G^1, G^2 and G^3 on which $A(G^1) = 1$ and $A(G^2) = A(G^3) = 0$, or we find a formula G without free variables that evaluates to 0 but $A(G) = 1$. In either case, we have found a set of at most three formulas on which A errs[1]. Note, finally, that this procedure is strongly non-black-box on A as it needs its description to build the formulas $G(A,n)$. The same idea with some additional technicalities gives the proof of (1) and this is what the authors of [6] did.

Let us now discuss the new proof of (1) that leads to a sampler that is essentially black-box. The new idea is to use the learning algorithm of Bshouty, Cleve, Gavaldà, Kannan and Tamon [3] that learns polynomial-size circuits in $\mathbf{ZPP}^{\mathbf{NP}}$ with the help of equivalence queries. The point is that if B is an algorithm that supposedly solves SAT, we may use it both to answer the equivalence queries, and to simulate the oracle in $\mathbf{NP}$, while trying to learn circuits for SAT itself. The argument now goes roughly as follows:

Assume $\mathbf{NP} \not\subseteq \mathbf{BPP}$ and let B be any $\mathbf{BPP}$ algorithm that supposedly solves SAT and that we wish to fool. For a given formula-size n, we run the learning algorithm but use B to answer the equivalence queries against SAT at length n, and also to simulate the oracle in $\mathbf{NP}$. To be more precise, the equivalence queries against SAT are simulated by querying B about the statement $S(C,n)$, where C is the circuit for which we ask the equivalence query, and $S(C,n)$ is the statement in the [6]-proof above. A refined analysis of the learning algorithm reveals that the result is, either (i) a small set of formulas on which B errs, namely those that simulate the equivalence queries, or their counterexamples, or the queries to the $\mathbf{NP}$-oracle, or (ii) a polynomial-size circuit for SAT. If the former case happens infinitely often, we are done[2]. Otherwise, we reach a contradiction because the learned circuits can now be used to solve SAT and thus collapse $\mathbf{NP}$ to $\mathbf{BPP}$. An analysis of the learning algorithm very similar to the one we need was observed before by Fortnow, Pavan and Sengupta [5], but their goal was quite different. The authors in [5] ask for further applications of this observation. This may be another. We discuss the small differences in Section 3.

Modulo the details, it seems clear that our new proof of the main result in [6] provides a distribution that is computable with black-box queries to B. Indeed, the only use we make of B is to answer the equivalence queries by querying it about a formula of the from $G(C,n)$ for a circuit C, or to answer the queries to the $\mathbf{NP}$-oracle that have nothing to do with B. Thus, it is not surprising that the argument extends easily to circuits under the assumption $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. This achieves our goal.

Discussion about this and previous work Classical average-case complexity theory considers $\mathbf{NP}$ problems with a fixed samplable distribution, or as we say, *distributional problems*. A distributional problem is average-case hard if every efficient algorithm has non-negligible probability of error when the probability is taken over the fixed distribution. Thus, the setting is very different from ours since our distributions may depend on the algorithm.

A worst-case to average-case reduction is a mapping from a language A to a distributional problem B, such that every algorithm M that has good probability of success on B can be transformed into an algorithm that solves A in the

[1] Note that the sizes of the formulas we found may not be n, but a clever trick devised in [6] takes care of this.

[2] Note again that the size of the formulas may not be n. But the same trick as before will work.

worst-case. The issue of worst-case to average-case reductions may be as old as cryptography itself. We suggest the introduction in [2] for a discussion. The reason we mention this topic here is because there are results showing that certain worst-case to average-case reductions are impossible if the reducing algorithm is black-box. Viola [9] proves so if the reducing algorithm is black-box both in the algorithm M and in the hard language A. Extending results by Feigenbaum and Fortnow [4], Bogdanov and Trevisan [2] show that, unless the polynomial-time hierarchy collapses, if A is $\mathbf{NP}$-complete, the reducing algorithm cannot be black-box on M if it makes non-adaptive queries only.

The 'pseudo'-version of average-case analysis, where the distribution may depend on the algorithm, has been studied to a much lesser extent. Several issues are still not very well understood, and some definitions are not totally stable yet. In this respect, it is fair to stress that our formal definition of black-box adversary has a component that may be considered not totally black-box. Namely, the adversary needs to know the running time of the algorithm it wants to fool, or the size of the circuits it wants to fool. This set aside, the adversary has absolutely no idea of what the algorithm does, and in this sense it is *black-box*. But the most important feature of the new definition is that it makes sense for circuits because a uniform adversary cannot have a non-uniform algorithm built-in in its code.

Finally, let us point out that the results in [6] may be used to prove our black-box version of (1). Indeed, if the adversary knows that the running time of the algorithm is n^k ahead of time, it can generate a distribution that is hard for all algorithms of time n^k simultaneously by choosing the code of such an algorithm at random and simulating its adversary (this requires to have adversaries for machines that do not conform to any error-gap; see [6, Theorem 2] for details). It is clear, however, that the same technique fails badly for proving

$$\mathbf{NP} \not\subseteq \mathbf{P/poly} \text{ implies } \mathbf{NP} \not\subseteq \textbf{bb-pseudo-}\mathbf{P/poly}$$

because non-uniform algorithms do not have small descriptions. Thus, the two proofs of (1), the one in [6] and the new one here, are really different.

2 Preliminaries

All our languages are over the binary alphabet $\{0, 1\}$. For a language A, we write $A(x)$ for the characteristic function of A. Thus, if x is a string in A, then $A(x) = 1$; and if x is a string not in A, then $A(x) = 0$. All our algorithms are modelled by Turing machines. We assume some familiarity with the basic concepts of complexity theory (see [1, 8]). Still, we review some. A language A belongs to $\mathbf{BPP}$ if there exists a probabilistic polynomial-time algorithm M such that, for every $x \in A$, we have $\Pr[M(x) = 1] \geq 2/3$, and for every $x \notin A$, we have $\Pr[M(x) = 1] \leq 1/3$. We say that M is a $\mathbf{BPP}$-algorithm for A. A language A belongs to $\mathbf{RP}$ if there exists a probabilistic polynomial-time algorithm M such that, for every $x \in A$, we have $\Pr[M(x) = 1] \geq 1/2$, and for every $x \notin A$, we have $\Pr[M(x) = 1] = 0$. A language A belongs to $\mathbf{P/poly}$ if there exists a family of Boolean circuits $(C_1, C_2, \ldots)$ and a polynomial $p(n)$ such that, for every n, the size of C_n is bounded by $p(n)$, and C_n computes the characteristic function of $A \cap \{0, 1\}^n$. It is known that $\mathbf{BPP} \subseteq \mathbf{P/poly}$. The proof of this inclusion is known as *Adleman's argument*. More precisely, if A is a language in $\mathbf{BPP}$ with a $\mathbf{BPP}$-algorithm M running in time $p(n)$, then Adleman's argument shows that A has Boolean circuits of size $O(p(n)^3)$.

Distributions and samplers We use U_n to denote the uniform probability distribution on $\{0, 1\}^n$. For every n, let E_n be a probability distribution on $\{0, 1\}^n$. We write $E = E_n$ to denote the *ensemble of distributions* $E = \{E_n\}_{n\geq 1}$. Let D be a probabilistic algorithm that takes 1^n as input and returns a string of length n as output. Each such algorithm defines a probability distribution D_n on $\{0, 1\}^n$ in the natural way: the probability that D_n assigns to $y \in \{0, 1\}^n$ is precisely the probability that D generates y on input 1^n. We say that a distribution $E = E_n$ is *polynomially samplable* if there exists a probabilistic polynomial-time algorithm D as above such that $D_n = E_n$ for every n. We say that the algorithm D is a *polynomial-time sampler*. The definition extends naturally to probabilistic oracle algorithms $D^?$. If D^O runs in polynomial time for every oracle O we say that $D^?$ is a *polynomial-time oracle sampler*.

Pseudo classes The concept of *pseudo* complexity classes was introduced by Kabanets [7] to model easiness against uniform adversaries. The idea is that a language A belongs to the pseudo-version **pseudo**-$\mathcal{C}$ of a complexity class $\mathcal{C}$, if there exists a language B in $\mathcal{C}$ that is indistinguishable from A by polynomially samplable distributions. Formally:

Definition 1 (Pseudo-classes, Kabanets [7]) *Let $\mathcal{C}$ be a class of languages, let A be a language. We say that A belongs to* **pseudo**-$\mathcal{C}$ *iff there exists a language B in $\mathcal{C}$ such that for every polynomial-time sampler D and every polynomial $p(n)$ we have $\Pr_{x\in D_n}[A(x) = B(x)] \geq 1 - 1/p(n)$ for all but finitely many n.*

Let us note here that the definition of $\textbf{pseudo-}\mathbf{BPP}$ used by Gotfreund, Shaltiel and Ta-Shma differs from the definition of Kabanets in one important aspect: the definition in [6] is stated in terms of the class of probabilistic polynomial-time algorithms. In other words, a language A belongs to $\textbf{pseudo}_\epsilon\textbf{-}\mathbf{BPP}$, in the definition of [6], if there exists a probabilistic polynomial-time algorithm B such that for every polynomially samplable distribution $D = D_n$ we have $\Pr_{x \in D_n}[A(x) = B(x)] \geq 1 - \epsilon$ where the probability is both over x *and* the internal coin-tosses of B. Since our goal is to extend the results in [6] to black-box adversaries, we will have to stick to the language view.

We define now the black-box versions of **pseudo-$\mathbf{BPP}$** and **pseudo-$\mathbf{P/poly}$**. As we discussed in the introduction, our adversary will be allowed to make oracle calls to B only, but he may know the running time of a $\mathbf{BPP}$-algorithm for B. We formalize this by giving two inputs to an oracle sampler $D^?$: the input-length n, and the running time n^k of the $\mathbf{BPP}$-algorithm.

Definition 2 *A language A belongs to* **bb-pseudo-$\mathbf{BPP}$** *iff for every polynomial-time oracle sampler $D^?$, there exists a language B in $\mathbf{BPP}$ with a $\mathbf{BPP}$-algorithm running in time n^k for some k, such that for every polynomial $p(n)$ we have $\Pr_{x \in D_n}[A(x) = B(x)] \geq 1 - 1/p(n)$ for all but finitely many n, where D_n is the probability distribution generated by $D^B(1^n, 1^{n^k})$.*

For $\mathbf{P/poly}$, the definition is similar where the running time is replaced by the size of the circuits.

Definition 3 *A language A belongs to* **bb-pseudo-$\mathbf{P/poly}$** *iff for every polynomial-time oracle sampler $D^?$, there exists a language B in $\mathbf{P/poly}$ with circuits for size n^k for some k, such that for every polynomial $p(n)$ we have $\Pr_{x \in D_n}[A(x) = B(x)] \geq 1 - 1/p(n)$ for all but finitely many n, where D_n is the probability distribution generated by $D^B(1^n, 1^{n^k})$.*

Let us also recall the i.o. quantifiers. Let $\mathcal{C}$ be a class of languages. We say that A belongs to **i.o.-$\mathcal{C}$** if there exists a language B in $\mathcal{C}$ and there exist infinitely many n such that $A \cap \{0,1\}^n = B \cap \{0,1\}^n$.

3 Learning circuits

The proof of our main result builds on the celebrated learning algorithm of Bshouty et al. for learning circuits. We discuss this algorithm in this section. We also state and prove the main property we need about it.

We say that an algorithm A is *given access to an equivalence-oracle with respect to a Boolean function $f : \{0,1\}^n \to \{0,1\}$* if A has the right to pose queries of the form: 'Does circuit D compute f?' The equivalence-oracle answers faithfully either 'yes' or 'no' in unit time, and in case it answers 'no' it also provides a *counterexample* x such that $D(x) \neq f(x)$. The result can be stated as follows:

Theorem 1 (Bshouty et al. [3]) *There exists a probabilistic polynomial-time oracle algorithm* LEARN *such that, for every circuit C with n inputs and size s, if* LEARN *is given access to an $\mathbf{NP}$-oracle and to an equivalence-oracle with respect to the Boolean function computed by C, then* LEARN$(1^n, 1^s)$ *returns a circuit that is equivalent to C with probability at least $3/4$, and returns 'don't know' with probability at most $1/4$.*

The key question that will lead us to the proof is the following: what happens to the learning algorithm LEARN$(1^n, 1^s)$ if it is given access to an equivalence-oracle with respect to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ that does not have circuits of size s? It is not too hard to see, from the proof of Theorem 1, that in this case, with probability at least $3/4$, either the algorithm returns a circuit for f of size bigger than s, or the collection T of all counterexamples generated by the equivalence-oracle is such that for every circuit C of size s there exists an $x \in T$ such that $C(x) \neq f(x)$. The reason for this is that LEARN is essentially an *approximate halving algorithm* that shrinks the space of consistent size-s circuits by a constant factor at each round.

A property very similar to the one we need was also observed by Fortnow, Pavan and Sengupta [5] who gave it a completely different use. The difference between the two properties is on the assumption: in [5] they assume that f does not have circuits of size $s^{O(1)}$ and conclude that the collection of counterexamples rules out every size-s circuit. We do not make any assumption and conclude that, with probability at least $3/4$, either the algorithm returns a good circuit of size at most $s \cdot n^2$, or the counterexamples rule out every size-s circuit. Our property is only slightly stronger and has essentially the same proof. Let us encapsulate the exact observation that we need in a lemma:

Lemma 1 *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and let s be an integer. Then, if* LEARN *is given access to an $\mathbf{NP}$ oracle and to an equivalence-oracle with respect to f, and if T denotes the collection of all counterexamples generated by the equivalence-oracle in an execution of* LEARN$(1^n, 1^s)$*, then, with probability at least $3/4$ over the internal coin-tosses, either* LEARN$(1^n, 1^s)$ *returns a circuit for f of size at most $s \cdot n^2$, or for every circuit C of size s, there exists an $x \in T$ such that $C(x) \neq f(x)$.*

Proof sketch: The algorithm LEARN produces a collection S of pairs $(x, f(x))$ inductively as follows. Initially, $S_0 = \emptyset$. Having built $S_i = \{(x_1, f(x_1)), \ldots, (x_i, f(x_i))\}$, the algorithm proceeds to extend it by one more pair. Using the $\mathbf{NP}$-oracle and its randomness, LEARN samples a collection of circuits $D_1, \ldots, D_q$, each of size s, independently and approximately uniformly at random from the collection of all circuits of size s that are consistent with the current

S_i. Then it forms the circuit D that combines $D_1, \dots, D_q$ by putting a majority gate on top, and queries it to the equivalence oracle. If the equivalence-oracle says that D computes f, the algorithm stops. Otherwise, the equivalence-oracle provides a counterexample x_{i+1}. The crux of the argument is that $D_1, \dots, D_q$ are approximately uniformly distributed among the circuits of size s that are consistent with S_i, so a counterexample to D is very likely to be a counterexample to a constant fraction of the circuits of size s. It follows that after $t = O(s)$ rounds, there is a good chance that either we produced a circuit for f, or there are no size-s circuits that are consistent with S_t (see [3] and [5] for details). □

4 Alternative and stronger proof

In this section we present an alternative proof to the main result in [6]. In fact, our argument proves something stronger. This will lead to our main result about $\mathbf{P/poly}$. We start with the statement of the black-box version of the main result in [6], which is what our argument proves:

Theorem 2

$$\mathbf{NP} \not\subseteq \mathbf{BPP} \ \textit{implies} \ \mathbf{NP} \not\subseteq \textbf{bb-pseudo-BPP}.$$

Before we get into the proof, we need some preparations. For a circuit D with n inputs, let $S(D)$ be the following statement:

> *There exists a Boolean formula F of size n such that, either $D(F) = 0$ and F is satisfiable, or $D(F) = 1$ and $D(F|_{x_1=0}) = D(F|_{x_1=1}) = 0$, or $D(F) = 1$ and F has no free variables and evaluates to 0.*

By an appropriate encoding of Boolean formulas, we may assume that the sizes of $F|_{x_1=a}$ and F are the same. One way to do this is by saving a couple of bits next to the encoding of each propositional variable of F to encode the fact that a variable has been instantiated or not, and by which value if so.

It is clear that $S(D)$ is decidable in $\mathbf{NP}$ when given D as input. By the Cook-Levin reduction, there exists a polynomial-time computable function g that, given a circuit D with n inputs and size s, returns a Boolean formula $g(D)$ that has size polynomial in n and s and is satisfiable if and only if $S(D)$ is true. Let $q(n, s)$ be the size of the formula returned by g when it is given a circuit with n inputs and size s. We may assume the size of $g(D)$ is the same for each such D.

Lemma 2 *Let D be a Boolean circuit with n inputs. Then, the following are equivalent:*

1. *$S(D)$ is true;*
2. *$g(D)$ is satisfiable;*
3. *D does not compute* SAT *on Boolean formulas of size n.*

Proof: The equivalence of (1) and (2) follows from the Cook-Levin reduction. Let us show the equivalence of (1) and (3). If D computes SAT on Boolean formulas of size n, it is clear that $S(D)$ is false. Conversely, if D does not compute SAT on Boolean formulas of size n, then either it returns 0 on a satisfiable formula F, or it returns 1 on an unsatisfiable formula F. In the former case, we are done because $S(D)$ is true. In the latter case, consider the value of D on $F|_{x_1=0}$ and $F|_{x_1=1}$. If both are 0, we are done because $S(D)$ is true. Otherwise, we set $F := F|_{x_1=a}$ for the one that gives 1 and recurse. Since F is unsatisfiable, both $F|_{x_1=0}$ and $F|_{x_1=1}$ are unsatisfiable as well. Now, if the recursion reaches a formula F in which all variables are set, necessarily F evaluates to 0 yet D returns 1 on F. Hence $S(D)$ is true. □

The following lemma is already very close to our goal of proving Theorem 2. If T is a set of formulas and B is language, we say that B *fails to solve* SAT *on* T when there exists a formula $x \in T$ such that $B(x) \neq \text{SAT}(x)$.

Lemma 3 *If* SAT $\notin \mathbf{BPP}$*, then there exists a probabilistic polynomial-time oracle algorithm $A^?$ such that, for every language B in $\mathbf{BPP}$ with a $\mathbf{BPP}$-algorithm running in time n^k, there exist infinitely many n such that the algorithm A^B on input 1^n and 1^{n^k} returns, with probability at least $1/8$, a set T of Boolean formulas on which B fails to solve* SAT.

Proof: Let us assume SAT $\notin \mathbf{BPP}$. The algorithm $A^?$, on inputs 1^n and 1^{n^k}, starts simulating the algorithm LEARN on inputs 1^n and $1^{n^{3k}}$ with an equivalence-oracle for SAT. That is, the learning algorithm is allowed to ask queries of the form: 'Is D a circuit that computes SAT on formulas of size n.' Since this strong equivalence-oracle is not available, we will have to show how to simulate it using the oracle we do have. Similarly, the $\mathbf{NP}$ oracle that LEARN uses is not available, so we simulate it as well. The idea is that we pretend that the oracle we have correctly solves SAT despite we know it doesn't.

We may assume that the $\mathbf{NP}$-oracle that LEARN uses is SAT. Whenever LEARN asks its $\mathbf{NP}$-oracle a query q, we add q to the set T, and we answer the query by using the oracle we have instead of SAT. This is all for the simulation of the $\mathbf{NP}$-oracle. Note that the oracle we have may be completely wrong on q, but we do not care and proceed with the simulation of LEARN anyway. Now we turn to the simulation of the equivalence-oracle. Whenever LEARN asks an equivalence query D, we compute the formula $G = g(D)$ expressing the statement $S(D)$ and query it to the oracle we have. If the answer to G is 0, then our oracle claims that G is unsatisfiable, so it claims that D computes SAT on formulas of size n. In this case we add G to T and proceed with the simulation of LEARN. On the other hand, if the answer to the query is 1, then our oracle claims that G is satisfiable, so it claims that D does not compute SAT on formulas of size n. Then we start the downward self-reducibility of SAT searching for a satisfying assignment to G but using our oracle instead of SAT. If the search succeeds, we have a Boolean formula F of size n witnessing $S(D)$. This means that F is a counterexample to the equivalence query D with respect to SAT and the simulation of LEARN can proceed. If the search does not succeed, the we produced a set of at most three formulas that constitutes a flagrant proof that our oracle does not compute SAT. We add them to T and stop.

After polynomially many steps, LEARN either produces a circuit D or says 'don't know'. In the first case we add $G = g(D)$ to T. In the latter, we add to T the collection of all counterexamples $F_1, \ldots, F_t$ that have been generated in the simulation. This concludes the construction of $A^?$. The correctness will follow from the next claim:

Claim 1 *Let B be a language in $\mathbf{BPP}$ with a $\mathbf{BPP}$-algorithm running in time n^k. Then, there exists infinitely many n such that $A^B(1^n, 1^{n^k})$ produces, with probability at least $1/8$, a set T on which B fails to solve* SAT.

Proof of claim: Suppose on the contrary that, for all but finitely many n, the probability that B fails to solve SAT on T is less than $1/8$. This means that the simulation of LEARN reaches the end with probability at least $7/8$ because whenever we stop the simulation prematurely is because we have found a flagrant proof that $B \neq$ SAT and we have added the witnesses to T. Note that when we reach the end, either we produce a circuit, or add the collection of all counterexamples to T. We claim that for all but finitely many n, with probability at least $5/8$, the simulation of LEARN produces

1. either a circuit for SAT on formulas of length n,
2. or a collection of formulas T on which every circuit of size n^{3k} fails to solve SAT.

Indeed, if all oracle answers were correct, the probability that both (1) and (2) fail would be bounded by $1/4$ by Lemma 1. Since the probability that some oracle answer is incorrect is bounded by $1/8$, it follows that both (1) and (2) fail with probability at most $1/8 + 1/4 = 3/8$.

Now, let us argue that there exist infinitely many n such that the probability of (1) is less than $1/2$. Otherwise we could solve SAT in $\mathbf{RP}$ as follows: given a formula F of size n, run $A^B(1^n, 1^{n^k})$, and take the resulting circuit, if any, to decide the satisfiability of F; if the answer is that F is satisfiable, run the downward self-reducibility of SAT to produce a satisfying assignment for F; if we succeed, we accept, otherwise, we reject. Assuming that the probability of (1) is at least $1/2$ for all but finitely many n, this is an $\mathbf{RP}^{\mathbf{BPP}}$ algorithm that solves SAT with error bounded by $1/2$ for all but finitely many n. It follows that SAT $\in \mathbf{BPP}$, which contradicts the hypothesis.

Hence, there exist infinitely many n for which the probability of (2) is at least $5/8 - 1/2 = 1/8$. But B has a $\mathbf{BPP}$-algorithm running in time n^k, so B has n^{3k}-size circuits by Adleman's argument. From this we conclude that there exist infinitely many n for which B differs from SAT on some formula in T with probability at least $1/8$. This contradicts the assumption and the proof is over. □ (of claim and of lemma)

A second look at the proof of Lemma 3 reveals that every formula in T is of one of the following types: either it is query q to the $\mathbf{NP}$-oracle (which we assumed to be SAT), or it is a formula of the form $g(D)$, where D is a Boolean circuit, or it is a counterexample to an equivalence query and is hence a formula of size n. By appropriate padding, we may assume that the formulas of the types q and $g(D)$ have exactly the same size. Let us say this size is n^{ck} for some constant c that depends only on the running time of LEARN$(1^n, 1^{n^k})$. This gives a collection of formulas T of sizes n or n^{ck}. We can also bound the size of T by the running time of the algorithm $A^?$. Let us say that the size of T is at most n^{dk} for some constant d that depends only on the running time of the algorithm LEARN. This will be useful for the next proof.

Proof of Theorem 2: Suppose SAT $\not\in \mathbf{BPP}$. We show that SAT $\not\in$ **bb-pseudo-BPP**. Let $A^?$ be the algorithm of Lemma 3. Consider the sampling algorithm $D^?$ that, on inputs 1^n and 1^{n^k}, runs $A^?(1^n, 1^{n^k})$ and $A^?(1^{n^{1/ck}}, 1^{n^{1/c}})$. This may produce a set of formulas of sizes $n^{1/ck}$, n, or n^{ck}. Let T be the collection of all such formulas that have size n. The cardinality of T is bounded by $2n^{dk}$. Then we choose one formula uniformly at random in this set and output it.

Let us now argue that this sampling algorithm does what we want. We follow essentially the same reasoning as in [6]. Let B be a language in $\mathbf{BPP}$ with a $\mathbf{BPP}$-algorithm running in time n^k. Let $p(n) = 32n^{dk}$. By Lemma 3, there exist infinitely many n for which the algorithm A^B on inputs 1^n and 1^{n^k} returns, with probability at least $1/8$, a set Q of formulas on which B fails to solve SAT. Call such n's useful. Consider now the following two events: (i) Q contains some formula of size n and (ii) Q contains some formula of size n^{ck}. On a useful n, at least one of these two events must have probability at least $1/16$. Therefore, for infinitely many n, the probability that either $A^B(1^n, 1^{n^k})$

or $A^B(1^{n^{1/ck}}, 1^{n^{1/c}})$ outputs a formula of size n on which B fails to solve SAT is at least $n^{-dk}/32$. Hence, there exist infinitely many lengths n, on which the sampler D^B fools B with probability at least $p(n)$. □

To close this section, let us point out that we did not make any effort to optimize the probability that the adversary will find a counterexample. We are satisfied with non-negligible probability. This should be contrasted with the results in [6] where a counterexample is produced with constant probability $3/100$. Let us also remark that the hypothesis $\mathbf{NP} \not\subseteq \mathbf{BPP}$ in Theorem 2 may be replaced by $\mathbf{NP} \not\subseteq \mathbf{RP}$. Indeed, by standard gap-amplification arguments and the downward self-reducibility of SAT, it is not hard to show that both conditions are equivalent.

5 Main result

The bulk of the argument for our main result was exposed already in the proof of Theorem 2. As a matter of fact, the proof is now even easier because we do not have to worry about the possibility that the learning algorithm may sometimes work! In other words, it is immediate that case (1) in Claim 1 cannot occur for almost all n.

Theorem 3

$\mathbf{NP} \not\subseteq \mathbf{P/poly}$ *implies* $\mathbf{NP} \not\subseteq$ **bb-pseudo-**$\mathbf{P/poly}$.

Proof: Assume SAT $\notin \mathbf{P/poly}$. Consider the oracle algorithm $A^?$ from Lemma 3. The new version of Claim 1 is the following:

Claim 2 *Let B be a language in $\mathbf{P/poly}$ with circuits of size n^k. Then, there exist infinitely many n on which $A^B(1^n, 1^{n^k})$ produces, with probability at least $1/8$, a set T on which B fails to solve* SAT.

Proof of claim: The proof proceeds as in Claim 1 until we argue that the probability of (1) cannot be at least $1/2$ for all but finitely many n. Something stronger is true here: the probability of (1) cannot be positive for all but finitely many n because SAT does not have polynomial-size circuits. It follows that there exist infinitely many n on which the probability of (2) is at least $5/8$; contradiction. □ (of claim)

A more careful analysis reveals that we can put $3/8$ instead of $1/8$ in Claim 2. Also, the algorithm $A^?$ on input 1^n and 1^{n^k} need not run LEARN$(1^n, 1^{n^{3k}})$ as LEARN$(1^n, 1^{n^k})$ suffices. The rest of the proof of Theorem 3 mimics the proof of Theorem 2, and we are done. □ (of theorem)

Next we would like to extend our results to i.o. classes and show that if $\mathbf{NP} \not\subseteq$ **i.o.-**$\mathbf{P/poly}$, then $\mathbf{NP} \not\subseteq$ **i.o.-bb-pseudo-**$\mathbf{P/poly}$. Unfortunately, the proof does not seem to go through directly. In fact, everything works fine except when we want to take care of the fact that the sampler from Lemma 3 may return formulas of different sizes. The trick we used was to consider the execution of $A^?$ on $1^n, 1^{n^k}$ and $1^{n^{1/ck}}, 1^{n^{1/c}}$ and keep only those formulas of size n that it produces, if any. The problem is that the hard instance it produces may be of sizes $n^{1/ck}$ or n^k, but not of size n. Perhaps a padding argument could fix this, but we do not see exactly how.

6 Conclusions and Open Problems

The most natural black-box version of Definition 1 is arguably the following:

Definition 4 (Revised black-box pseudo-classes) *Let $\mathcal{C}$ be a class of languages, let A be a language. We say that A belongs to* **strong-bb-pseudo-**$\mathcal{C}$ *iff for every polynomial-time oracle sampler $D^?$, there exists a language B in $\mathcal{C}$ such that for every polynomial $p(n)$ we have $\Pr_{x \in D_n^B}[A(x) = B(x)] \geq 1 - 1/p(n)$ for all but finitely many n.*

Note the important switch of quantifers between Definition 1 and Definition 4 and the absence of running times. We do not know if our results in Theorem 2 and Theorem 3 are true under this new notion, or if there is a good reason why this is not possible. It would be nice to understand these issues better. On the one hand, it is true that Definition 4 captures the pure essence of a black-box adversary. But on the other, providing the running time of the algorithm to the adversary is not giving him much information. This is particularly clear in the case of **bb-pseudo-**$\mathbf{P/poly}$ because there are exponentially many circuits of size n^k.

At the end of Section 5 we discussed the i.o. versions of our results and why the current proof does not seem to work. It ought to be possible to fix this by some sort of padding argument but we do not see how. A final issue we have not investigated is the analogue of these results for **co-NP** languages. Is it true that if SAT is not in **co-NP** then it is easy to find the hard instances to any **co-NP**-algorithm? This may be relevant for propositional proof complexity.

Acknowledgments: I want to thank Ricard Gavaldà for providing feedback at an early stage of this work, and the referees of CCC 2006 for the useful comments.

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, second edition, 1996.

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

[3] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–423, 1996.

[4] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal of Computing*, 22:994–1005, 1993.

[5] L. Fortnow, A. Pavan, and S. Sengupta. Proving SAT does not have Small Circuits with an Application to the Two-Queries Problem. In *18th IEEE Conference on Computational Complexity*, pages 347–357, 2003.

[6] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. If NP Languages are Hard on the Worst-Case Then It is Easy to Find Their Hard Instances. In *20th IEEE Conference on Computational Complexity*, pages 243–257, 2005.

[7] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 62(2):236–252, 2001. A preliminary version appeared in CCC 2000.

[8] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[9] E. Viola. The complexity of constructing pseudorandom generators from hard functions. *Journal of Computational Complexity*, 13(3-4):147–188, 2004. A prelinary version appeared in CCC 2003 under the title ‘Hardness vs. Randomness within Alternating Time’.