Journal of Computer and System Sciences **63**, 672–688 (2001)
doi:10.1006/jcss.2001.1780, available online at http://www.idealibrary.com on IDEAL®

Randomness vs Time: Derandomization under a Uniform Assumption

View metadata, citation and similar papers at core.ac.uk

Department of Computer Science, University of California—San Diego, San Diego, California 91097-0114
E-mail: russell@cs.ucsd.edu

and

Avi Wigderson[2]

Institute of Computer Science, Hebrew University, Jerusalem, Israel 91904
E-mail: avi@cs.huji.ac.il

Received May 1, 1999; revised April 1, 2001

We prove that if $BPP \neq EXP$, then every problem in BPP can be solved deterministically in subexponential time on almost every input (on every sampleable ensemble for infinitely many input sizes). This is the first derandomization result for BPP based on uniform, noncryptographic hardness assumptions. It implies the following gap in the deterministic average-case complexities of problems in BPP: either these complexities are always subexponential or they contain arbitrarily large exponential functions. We use a construction of a small "pseudorandom" set of strings from a "hard function" in EXP which is identical to that used in the analogous nonuniform results in L. Babai *et al.* (*Comput. Complexity* **3** (1993), 307–318) and N. Nisan and A. Wigderson (*J. Comput. System Sci.* **49** (1994), 149–167). However, previous proofs of correctness assume the "hard function" is not in $P/poly$. They give a non constructive argument that a circuit distinguishing the pseudorandom strings from truly random strings implies that a similarly sized circuit exists computing the "hard function." Our main technical contribution is to show that, if the "hard function" has certain properties, then this argument can be made constructive. © 2001 Elsevier Science (USA)

[1] Research Supported by NSF Award CCR-9734911 and CCR-0098197, Grant 93025 of the joint US–Czechoslovak Science and Technology Program, and USA–Israel BSF Grant 97-00188.

[2] This research was supported by Grant 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities.

© 2001 Elsevier Science (USA)
All rights reserved.

1. INTRODUCTION, HISTORY, AND INTUITION

1.1. Motivation

The introduction of randomization into efficient computation has been one of the most fertile and useful ideas in computer science. In cryptography and asynchronous computing, randomization makes possible tasks that are impossible to perform deterministically. For computing functions, there are many examples where randomization allows considerable savings in resources like space and time over any *known* deterministic algorithms. In other examples, the best deterministic algorithms are based on and considerably more complicated than the corresponding randomized algorithms.

But to what extent is this seeming power of randomness over determinism real? Is allowing randomization a powerful addition to the computational model, or is it just an algorithm design tool that can be eliminated afterwards? The most famous concrete version of this question regards the power of *BPP*, the class of problems solvable by probabilistic polynomial time algorithms making small constant error. What is the relative power of such algorithms compared to deterministic ones? This is largely open. On the one hand, it is possible that $P = BPP$, i.e., randomness is useless for solving new problems in polynomial time. On the other, we might have $BPP = EXP$, which would say that randomness would be a nearly omnipotent tool for algorithm design. *A priori*, neither extreme seems likely: there are some problems where randomness seems exponentially helpful, but many hard problems are not susceptible to randomized solutions.

In this paper, we show that the intuition that randomness is a resource basically incomparable to time is wrong. Either there is a non-trivial deterministic simulation of *BPP*, or $BPP = EXP$! Either time can non-trivially substitute for randomness, or randomness can non-trivially substitute for time. In other words, either universal derandomization is possible, or randomization is a panacea for intractability. (There are some technical provisos: the deterministic simulation only works for infinitely many input lengths, and may fail on a negligible fraction of inputs even of these lengths.) We consider the former much more plausible than the latter.

1.2. History: Hardness vs Randomness

While counter to most people's first intuition, our result should be less surprising to those who are aware of the literature on derandomization. The fundamental paradigm in derandomization is to trade "hardness" for "randomness." This was first elucidated in the remarkable sequence of papers [29, 10, 31]. Roughly speaking, "computationally hard" functions can be used to construct "efficient pseudorandom generators." These in turn lower the randomness requirements of any efficient probabilistic algorithm, allowing for a "nontrivial" deterministic simulation.

In many such results, there is a quantitative trade-off between the hardness assumption and the time to perform the deterministic simulation. The stronger the assumption, the faster the simulation. Here, we are concentrating on the "low end"

of the curve in this trade-off: what is the weakest assumption one can make and still have some version of universal derandomization? Our results also have some implications for the "higher end" of the curve. We will omit formal statements and proofs for these versions, but they are worked out in detail and applied to probabilistic time hierarchy results in [11].

We will thus compare our results mainly to the "low end" version of the known results. In particular, we will use as our standard for "nontrivial" the class $SUBEXP = \bigcap_{\delta>0} DTIME(2^{n^{\delta}})$. The statement $BPP \subset SUBEXP$ (read "randomness is weak"), while falling short of $P = BPP$, would be a great result to prove unconditionally, as it implies $BPP \neq EXP$. There have been a sequence of papers getting weaker and weaker hardness assumptions sufficient to prove such a result. These papers use one of three basic methods for converting hard functions into pseudorandom sequences: the "cryptographically secure" pseudorandom generator based on one-way functions ([10, 31, 22, 12, 13, 15]); the Nisan–Wigderson-generator (NW-generator) based on a Boolean function with no circuit that approximates it ([27, 7]); and the hitting set method ([1, 2, 3, 4]).

To state our results, we will need some notation for complexity classes. Let $Size(T(n))$ be the class of functions computable by circuit families where the number of gates in the circuit with n inputs is at most $T(n)$. For C a complexity class and $t(n)$ a function, let $C/t(n)$ be the class of functions computable in C with $t(n)$ bits of "advice" depending only on the input size, i.e., $f \in C/t(n) \Leftrightarrow \exists g \in C$ and a function $h: Z \to Z$ with $|h(n)| \leqslant t(n)$ and $f(x) = g(x, h(|x|))$. This notation was introduced in [21], who note that $P/poly = \bigcup_{c \geqslant 1} Size(n^c)$. For C a complexity class, let $i.o.-C$ be the class of functions that agree with a function in C for all inputs of length n for infinitely many n.

The first set of papers construct a pseudorandom generator from a one-way function. The pseudorandom generator quickly converts a small random string to a polynomially larger string that seems random in the following sense: Any adversary that can distinguish an output of this generator from a truly random string of the same length can be used to invert the function. A *BPP* algorithm that had a markedly different behavior using a pseudorandom sequence as its source of randomness than using a truly random sequence would be such an adversary. (Note that, to *construct* the adversary from the *BPP* algorithm, an input for which the pseudorandom generator fails must be found. This will be the main distinction between our average-case derandomization and the worst-case derandomization possible with non-uniform assumptions.) So if no such invertor exists, the deterministic algorithm that enumerates the multiset of outputs of the generator and simulates the *BPP* algorithm on each, taking the majority answer, would always be correct. Informally, this is stated as:

THEOREM A.1 [10, 31, 13, 12, 15]. *If there are one-way functions that cannot be inverted with a nonnegligible probability in $P/poly$, then $BPP \subset SUBEXP$.*

The NW-generator [26, 27, 7] considerably weakened the hardness assumption needed in the nonuniform setting. It achieves the same deterministic simulation of *BPP*, from any function in $EXP - P/poly$. A key observation is that, for

derandomization purposes, we only require that the generator be computable in exponential time, since we are going to use its outputs on all input sequences in the deterministic simulation.

THEOREM A.2 [26, 27, 7]. *If $EXP \not\subset P/poly$, then $BPP \subset i.o.-SUBEXP$.*

This was the best result known at the "low end" of the hardness vs. randomness curve.

There has also been a sequence of papers ([27, 1, 2, 20]) at the "high end" of the curve, where the desired goal is to obtain $P = BPP$ under the weakest possible assumption. The strongest result in this sequence is stated as follows (with E denoting $DTIME(2^{O(n)})$:

THEOREM A.3 [20]. *If $E \not\subset i.o.-SIZE(2^{o(n)})$, then $BPP = P$.*

1.3. Average-Case Derandomization under Uniform Assumptions

Both the high-end and low-end results above require nonuniform hardness assumptions, i.e., that the hard problems in question are hard for circuits. The reason for this, intuitively, is that if a function is hard uniformly but easy nonuniformly, there is some advice, or trapdoor, that makes computing the function easy. Even if this trapdoor is hard to find, there is no way to guarantee that a rare instance of the *BPP* problem does not code this trapdoor information. Thus, it seems difficult to obtain a worst-case guarantee for the simulation.

However, we might still be able to get an "average-case" simulation under a uniform assumption. In fact, what such a result would say is that it is infeasible to find inputs for which such a simulation fails. We need to be somewhat careful in defining average-case complexity of a problem. We actually want to examine the difficulty of the problem for "most instances" rather than the average of the difficulties. We also should say what kinds of errors the algorithm is allowed to make on the exceptional instances. An algorithm allowed to output mistakes on a small number of inputs will be called a "heuristic" for the problem, whereas an algorithm that simply fails to give an answer in the alloted time will be called an algorithm with a certain average-case performance. Here, we'll use somewhat ad hoc definitions of these concepts that have certain technical advantages for our setting, especially when making statements about "infinitely many" sizes.

DEFINITION 1 (PROBABILITY DISTRIBUTIONS AND ENSEMBLES). For a probability distribution ρ over a finite set S, we will use the notation $r \in_\rho S$ to denote an element of S selected randomly according to ρ. We will use U to generically denote the uniform distribution over S. A *probability ensemble* $\mu = \{\mu_n \mid n \in Z^+\}$ is a sequence of probability distributions on the set of strings of length n.

DEFINITION 2 (POLYNOMIALLY SAMPLEABLE DISTRIBUTIONS). The ensemble μ is *polynomially sampleable* if there is a polynomial p and a polynomial time computable function M so that if $r \in_U \{0,1\}^{p(n)}$, then $M(n, r)$ is distributed according to μ_n. As usual, we can extend this notion to allow M access to an oracle, in which case we say μ is polynomially sampleable given the oracle.

DEFINITION 3 (HEURISTIC AND AVERAGE TIME CLASSES). Let T and ε be functions of n. $HeurTIME_{\varepsilon(n)}(T(n))$ is the class of functions $f: \{0,1\}^* \to \{0,1\}$ so that there is an algorithm $A(x)$ running in deterministic time $T(|x|)$ with the following property. Informally, it is infeasible to find instances on which A errs. Formally, for every polynomially sampleable probability ensemble μ, $\forall n$, when choosing $x \in_{\mu_n} \{0,1\}^n$, $\text{Prob}[A(x) \neq f(x)] < \varepsilon(n)$.

$AvgTIME_{\varepsilon(n)}(T(n))$ is the class of functions $f: \{0,1\}^* \to \{0,1\}$ so that there is an algorithm $A(x)$ running in deterministic time $T(|x|)$ with the following property. Informally, it is infeasible to find instances on which A runs too long. Formally, $A(x) \in \{f(x), ?\}$ (so A never errs) and for all polynomially sampleable probability ensembles μ, every n, when choosing $x \in_{\mu_n} \{0,1\}^n$, $\text{Prob}[A(x) \neq f(x)] < \varepsilon(n)$.

Under cryptographic assumptions, the standard techniques give "average-case" simulations. From uniformly one-way functions, we can generate pseudorandom sequences that are hard to distinguish by probabilistic algorithms (as opposed to circuits). A statement of the resulting derandomization is:

THEOREM A.4 [10, 31, 13, 12, 15]. *If there are uniformly one-way functions, then for every $c > 0$, $BPP \subset Heur_{1/n^c} SUBEXP$ and $ZPP \subset Avg_{1/n^c} SUBEXP$.*

1.4. Our Results

The main result of this paper is a version of this theorem based on the much weaker assumption $BPP \neq EXP$.

THEOREM 5. *If $BPP \neq EXP$ then for every $c > 0$, $BPP \subset i.o.-Heur_{1/n^c} SUBEXP$ and $ZPP \subset i.o.-Avg_{1/n^c} SUBEXP$.*

We also give a sharp converse:

THEOREM 6. *There are functions in $EXP \cap P/poly$ that are not in $i.o.-HeurTIME_{1/3}(2^{o(n)})/o(n)$.*

COROLLARY 7. *If $BPP \subset i.o.-HeurTIME_{1/3}(2^{o(n)})/o(n)$, then $BPP \neq EXP$.*

COROLLARY 8. *If $BPP = EXP \cap P/poly$ then $BPP = EXP$.*

Summarizing, we get:

COROLLARY 9. *Exactly one of the following holds:*

(1) $BPP = EXP$

(2) $BPP \subset i.o.-HeurTIME_{1/n^c}(2^{n^\delta})$ *for every* $\delta > 0$ *and* $c > 0$.

This result is naturally interpreted as a gap theorem on derandomization: either no derandomization of BPP is possible at all, or otherwise a highly nontrivial derandomization is possible. A more precise statement of the gap is:

COROLLARY 10. *If $BPP \subset i.o.-HeurTime_{1/3}(2^n)/o(n)$ then $BPP \subset i.o.-HeurTIME_{1/n^c}(2^{n^\delta})$ for every $\delta > 0$ and $c > 0$.*

1.5. Summary

Each of the above simulations of *BPP*, including this paper, construct a version of pseudorandom sequence set, S. The difference between the cryptographic and non-cryptographic versions are whether it is possible to sample from S in significantly less time than it takes to enumerate the members of S. The difference between nonuniform and uniform pseudorandom sets is whether S looks random to a circuit or to a probabilistic algorithm. Finally, there are "high-end" results, where based on a strong assumption, S is close to polynomial size, and "low-end" results, where based on a weaker assumption, S is sub-exponential. This is summarized in the Fig. 1.

A few classifications of pseudo-random generators were omitted from the above chart. In [18, 19], the authors give middle level, non-uniform, non-cryptographic constructions, but the results are a bit too complicated to fit easily. A high-end, uniform, non-cryptographic construction is, to our knowledge, still an open problem.

1.6. Why Wasn't This Paper Written in 1988 (but Could Have Been)?

The rest of this section describes intuitively the obstacles to obtaining this result long ago, and the key ideas we use to overcome them. We stress again that these ideas are mostly viewing the known previous results from the "right" viewpoint, and hardly any additional technical work is required beyond stating them.

Attempts to find a uniform version of Theorem A2 (namely, replacing $P/poly$ by *BPP* in the hardness assumption) followed immediately after its discovery in 1988, both by the authors and many others. However, the following presented a psychological barrier.

The proofs of the afore-mentioned theorems have the following structure. A (presumably hard) function f is used to construct a (hopefully pseudorandom) generator G. Then, the proof assumes a hypothetical distinguisher for G, and constructs from it an efficient algorithm for f, obtaining a contradiction. In the nonuniform versions the distinguisher and algorithm are circuits, and only an *existence* proof of the algorithm from the distinuisher is required. In the uniform case, both are probabilistic Turing machines, and so an *efficient* construction of the algorithm from the distinguisher is needed.

The construction of an algorithm for f from a distinguisher of G, follows a sequence of steps, each of them of a similar form (constructing one algorithm with a given property assuming another algorithm with another property). Various steps in this construction seem to inherently need high computational complexity. In particular, they need values of f at many (often random) points. While this is easy if f was the inverse of a one-way function (these points can be obtained by computing in the easy direction) or nonuniformly (simply hardwire these values), such values seem impossible to obtain uniformly for an arbitrary function in *EXP*.

Here we take a careful look at the sequence of steps (of constructing one algorithm from another), which compose to give the algorithm for f from the distinguisher of G. We feel that formalizing these "reductions between constructions of

Classification	Size of Set	Sample Time	Weakest Assumption	Consequence
Non-Uniform Crypto, low-end	subexp	poly	one-way func. vs poly circuits [HILL]	$BPP \subset SUBEXP$
Non-Uniform Crypto, middle	quasi-poly	quasi-linear	one-way func. vs subexp circ [HILL]	$BPP \subset QuasiP$
Non-Uniform Crypto, high end	poly	quasi-linear	one-way perm. vs. exp circ. [GL]	$BPP = P$
Uniform Crypto, low end	subexp	poly	one-way func. vs poly time [HILL]	$BPP \subset HeurSUBEXP$ $ZPP \subset AvgSUBEXP$
Uniform Crypto, middle	quasi-poly	quasi-linear	one-way func. vs subexp time [HILL]	$BPP \subset HeurQuasiP$ $ZPP \subset AvgQuasiP$
Uniform Crypto, high end	poly	quasi-linear	one-way perm. vs exp. time [GL]	$BPP \subset HeurP$ $ZPP \subset AvgP$
Non-Uniform Non-Crypto low end	subexp	subexp	$EXP \not\subset P/poly$ [BFNW]	$BPP \subset i.o. - SUBEXP$
Non-Uniform Non-Crypto, high end	poly	poly	$E \not\subset io - SIZE(2^{o(n)})$ [IW97]	$BPP = P$
Uniform Non-Crypto low end	Sub-Exp	SUBEXP	$EXP \not\subset BPP$ This paper	$BPP \subset i.o - HeurSUBEXP$ $ZPP \subset i.o. - AvgSUBEXP$
Uniform Non-Crypto, middle	quasi-poly	quasi-poly	$\#P \not\subset BP(SUBEXP)$ [CNS]	$BPP \subset i.o. - HeurQuasiP$ $ZPP \subset i.o. - AvgQuasiP$

FIG. 1. A summary of constructions of pseudorandom sets of sequences. Here, exp means $2^{\Omega(n)}$, subexp is $2^{n^{o(1)}}$, quasi-polynomial is $n^{(\log n)^{O(1)}}$, and quasi-linear is $n(\log n)^{O(1)}$. Most results were obtained by long chains of papers; only the last in the sequence is cited.

algorithms/circuits," done in Section 2.2, and pinpointing their computational needs, is already an important psychological step.

We first note that for some of these steps (Random Self Reducibility [25, 8, 6], Hard Core Bit theorem [13]) the construction was already given uniformly (in *PPT*—probabilistic polynomial time) in the original papers. More importantly we observe that in the other (computationally harder) steps (the XOR Lemma [31], the generator conversion [26, 27]), function values of f are the *only* nonuniform

construct needed. Thus, the first key idea is allowing our uniform PPT algorithm to have an oracle for f.

At first sight it seems ridiculous to give an algorithm trying to compute f an access to an oracle for f. However, we don't merely want to compute f on a single input, but to construct a circuit computing f on all inputs. Thus, one could state the issue of whether such a construction exists as whether f is *learnable from examples* in the sense of computational learning theory (in an "active learning" model). With this observation (all technical work was already done!), we complete a proof that a distinguisher for the pseudorandom generator can be used to *uniformly* learn a circuit computing the hard function from examples.

So we get the following (informal) partial result (which already is very nontrivial!): if an NW-generator based on any $f \in EXP$ is not pseudorandom against uniform algorithms, then a circuit for f can be constructed in PPT^f.

However, we promised you more! We still need to convert this into a construction of such a circuit with no oracle calls. This is not trivial, and in general probably impossible. However, the next crucial observation is that in the construction above the use of the oracle is limited in the following way: it is never called on larger input lengths than those of the circuit it constructs. How can this help to eliminate the oracle?

The next key idea is assuming (for no good reason so far) that f happens to be downward self reducible (like SAT or PERMANENT). In such cases observe that the oracle for f is redundant, since we can construct circuits for all input length inductively. To construct a circuit for f_n, simply use the above PPT algorithm, and whenever it calls the oracle, use the downward self reduction and the inductively constructed circuits of smaller sizes.

So now using e.g., the Permanent function we get the following (informal) partial result (which again is very nontrivial): if an NW-generator based on any $f \in \#P$ is not pseudorandom, then a circuit for f can be constructed in PPT, and so $f \in BPP$. In other words, the weaker assumption $\#P \neq BPP$ already gives us the required nontrivial derandomization.

But we promised you more! For that we need a complete problem for EXP that is downwards self-reducible. This seems impossible, since downward self-reducible problems are always in $PSPACE$. To see that no contradiction arises, we use at this point the known *nonuniform* derandomization results! We know from [7] that the only way the NW generator fails is if $EXP \subset P/poly$. But then it follows from [21] and [30] that $EXP = \Sigma_2 = P^{\#P} = PSPACE$.[3]

So if the NW generator fails with a standard EXP-complete problem, we try again with a downward self-reducible $\#P$-complete problem—a variant of the Permanent (which we then know is also EXP-complete). If the derandomization still fails, it now gives a BPP algorithm $\#P$ complete problem, and hence any problem in EXP. It is not surprising that the function we'll need has to be random self-reducible as well, but luckily Permanent has this property too!

[3] The result from [21] does not relativize [17], which suggest that similar methods might lead to non-relativizing separations. However, we are unsure whether our main result relativizes.

2. PROOF OF THEOREM 5

2.1. Overview

The proof will be by contradiction. We first describe the algorithm for every *BPP* (resp. *ZPP*) language. It will use, as usual, a hard function (assumed not in *BPP*) in the NW-generator. If this algorithm fails to have the desired properties, we'll derive a distinguisher for the generator and from it a *BPP* algorithm for f, contradicting the assumption.

• *The hard function.* We will use any Σ_2-hard Boolean function f in *EXP* that has the desired random self-reducibility and downward self-reducibility. In Section 2.3 we define these properties formally. The main (simple) result (Lemma 12) of this section is that such functions exist—a variant of the permanent is one.

• *The generator.* In Section 2.4 we will (essentially, with minor differences) follow the steps in the construction in [27, 7] of a generator G from the above function f. (In fact, there will be a sequence of generators, as we need an arbitrary polynomial stretch of the initial seed.) The main result of this section is Lemma 14, stating that given a distinguisher to G, a circuit for f_n can be efficiently computed with oracle access to f_n.

• *The derandomization.* In Section 2.5, as is standard since Yao's paper, for every language $L \in BPP$ (resp. *ZPP*), we fix a probabilistic algorithm for L. For every input, rather than flipping coins, we run deterministically over all possible outputs of the generator G of the appropriate length, and take a majority vote of the answers of the algorithm on each. The main result of this section (Lemma 18) is that if this algorithm fails to be in Heuristic (resp. Average) subexponential time, then we have an efficient distinguisher.

• *Removing the oracle.* In Section 2.6, we start by noticing that combining the above result, if the conclusion of the theorem holds, we have two things. First, we have an efficient learning algorithm for f_n given an oracle for f_n. Second, we know (from nonuniform derandomization) that $EXP = \Sigma_2$, and so f is complete for *EXP*. Combining these, we can learn a downward self-reducible function in the above manner. The main result of this section (Lemma 19) shows that the oracle is not needed in this learning algorithm. Thus it is a *BPP* algorithm for f, contradicting our assumption.

2.2. Reductions between Construction Problems

As mentioned above, the essence of this work is a careful view of the computational aspects needed by various steps of the hardness to randomness conversion. This section develops a convenient notation for the particular reductions these steps accomplish, namely reductions among circuit construction problems. It will also give rise to a (somewhat nonstandard) definition of random self reducibility. We also include (at the end) the standard Turing reduction, and use it to define downward self-reducibility.

Most of the algorithms we use are probabilistic polynomial time algorithms whose inputs and outputs will be encodings of Boolean circuits. We'll be interested in statements, "If one can construct a circuit with property X, then from it one can construct a circuit with property Y." The notation will be more general, but for our paper, one can think of most construction problems as specifying a type of circuit, and one can usually think of n as the number of inputs to these circuits.

DEFINITION 4. A *construction problem* $A = \{A_n\}$ is a family of nonempty subsets $A_n \subset \{0,1\}^*$. (Note that no upper bound is put on the sizes of members of A_n in terms of n.)

In the following definitions, the probabilities are taken over the uniform distribution on n bit strings.

DEFINITION 5. IMPORTANT CONSTRUCTION PROBLEMS.

- **Circuits approximating** f

Let $f: \{0,1\}^* \to \{0,1\}^*$ and $\varepsilon: N \to [0,1]$. Define the construction problem $C^{f,\varepsilon}$ as follows: $C_n^{f,\varepsilon}$ contains all circuits C with n inputs satisfying

$$\Pr_{x \in_U \{0,1\}^n} [C(x) = f(x)] \geqslant \varepsilon(n)$$

- **Circuits computing** f $C^f = C^{f,1}$.
- **Distinguishers** Let $m: N \to N$, $G = \{G_n: \{0,1\}^{m(n)} \to \{0,1\}^n\}$, and ε as before. Define the construction problem $D^{G,\varepsilon}$ as follows: $D_n^{G,\varepsilon}$ contains all circuits D with n inputs satisfying $\Pr_{y \in_U \{0,1\}^{m(n)}} [D(G(y)) = 1] - \Pr_{x \in_U \{0,1\}^n} [D(x) = 1] \geqslant \varepsilon$.

DEFINITION 6. Let A and B be construction problems.

A *efficient construction* for A is, intuitively, an efficient algorithm that generates a member of A with arbitrary high probability. Formally, it is a probabilistic algorithm taking two inputs, n, α, which runs in time $poly(n/\alpha)$, and whose output is a member of A_n with probability at least $1-\alpha$ (over the algorithm's coin tosses).

An *efficient construction of B from A*, is a probabilistic polynomial time algorithm which for every n, $\alpha > 0$, $a \in A_n$, outputs a member of B_n with probability at least $1-\alpha$. If such a construction exists, we denote it $A \to B$.

The following observation is immediate from the definitions:

LEMMA 11. *The relation* $\to$ *is transitive. Moreover, if* $A \to B$ *and* A *is efficiently constructible, then* B *is efficiently constructible.*

Finally, as usual, we can extend these definition by allowing the function f access to an oracle, O, which we will write $A \to^O B$. If, in addition, the queries made by the construction are all of binary length $m(n)$, we write $A \to^{O_{m(n)}} B$. The natural analogs of the lemma above hold as well.

2.3. Our Hard Function and Its Properties

This section describes the properties we require of our "hard" function, and present one that has them.

We will only require our function f to have the following weak kind of random self-reducibility. This weak random self-reducibility is clearly implied by all the usual definitions:

DEFINITION 7. A function $f: \{0,1\}^* \to \{0,1\}^*$ is weakly random self-reducible (*WRSR*) if $(C^{f,1-n^{-c}} \to C^f)$ for some $c \geqslant 0$.

The next property we'll need is downward self-reduction. To define it, we need the standard notion of Turing reducibility, and as in the previous subsection we carefully consider the input length of oracle queries.

DEFINITION 8. Let $f, g: \{0,1\}^* \to \{0,1\}^*$, and let $\ell: Z^+ \to Z^+$. We say that f is *polynomial Turing reducible to g restricted to length* $\ell(n)$, and write $f_n \leqslant_T^p g_{\ell(n)}$, if there is a deterministic[4] polynomial time oracle machine M^g that on every input x outputs $f(x)$ and queries the oracle g only on inputs of length at most $\ell(|x|)$.

DEFINITION 9. f is *downward self-reducible* **(DSR)** if $f_n \leqslant_T^p f_{n-1}$.

The last property we'll need is that the function is at least Σ_2 hard.

Now we are ready to present a function with these properties. Let *ModPerm* be the following decision problem:

Instance. An integer k in unary, a prime $p > 2k$ in unary, a $k \times k$ matrix M of integers modulo p, and an integer t modulo p.

Problem. Is $Perm(M) \bmod p = t$?, where *Perm* is the usual permanent function.

Note that it is easy to generate random valid instances of *ModPerm* of a given length, and to place them in one-to-one correspondence with integers up to the number of valid instances. So without loss of generality, by "uniform" distribution on length n strings for *ModPerm*, we mean uniform on the valid instances of length n. *ModPerm* is downward self-reducible by the usual method of computing permanent via minors. By using the Chinese Remainder Theorem, it is easy to see that computing the permanent for an arbitrary matrix of integers reduces to *ModPerm*, so it is complete for $\#P$. The self-reducibility for permanent modulo a prime $> 2k$ due to Lipton [25] (using the method of [8]) shows that *ModPerm* is *WRSR*. In the sequel, the reader can think of f as *ModPerm*, although we prefer to state things in more general terms.

LEMMA 12. *There is a #P-complete decision problem f that is DSR and WRSR.*

2.4. Construction of G from f

We view the construction of G from f as a sequence of three steps, in order to make the proof more modular. The sequence we use in the following is not exactly the one used in [27, 7], but the original one would work as well. As we'll need arbitrary polynomial stretch of randomness, we parameterized the generators by a constant d.

[4] A probabilistic version can be given and used as well, but we shall not need it.

Let $d > 0$ be any integer—it will specify the output length of our generator G_d. Let $c > 0$ be the constant so that $C^f \to C^{1-n^{-c},f}$.

Direct product function. Let $n_1 = n^{c+2}$. View an n_1 bit string as n^{c+1} n bit strings. Define g: $\{0,1\}^{n_1} \to \{0,1\}^{n^{c+1}}$ by $g(x_1, \ldots, x_n) = f(x_1), \ldots f(x_n)$.

Hard-core bit. Let $n_2 = n_1 + n^{c+1}$. View an n_2 bit string as an input x to g and a string r of length $|g(x)|$. Then $h(x, r) = \langle g(x), r\rangle$, where $\langle y, z\rangle$ represents inner product modulo 2.

Almost disjoint sets generator. Let $m = n_2^2$, and $\ell = n^d$. Let $z \in \{0,1\}^m$ and let $S = \{s_1 < s_2 < \cdots < s_{n2}\}$ be a subset of bit positions between 1 to m. Then define $z|_S$ to be the n_2 bit string $z_{s_1} z_{s_2} \cdots z_{s_{n_2}}$. In [27], an explicit construction of ℓ such sets $S_1, \ldots, S_\ell$ is given so that $|S_i \cap S_j| \leqslant \log_n \ell = d$ for every $i \neq j$. We define $G_d^f: \{0,1\}^m \to \{0,1\}^\ell$ by $G_d^f(z) = h(z|_{S_1}), h(z|_{S_2}), \ldots, h(z|_{S_\ell})$.

From the construction, it is clear that $G_d^{f_n} \leqslant_T^p h_{n_2} \leqslant_T^p g_{n_1} \leqslant_T^p f_n$, which proves the following Lemma 13.

LEMMA 13. $G_d^{f_n} \leqslant_T^p f_n$.

We will use this towards our main result of this section, showing that a distinguisher for $G_d^{f_n}$ can be used in conjunction with an oracle for f_n to efficiently construct a (small) circuit for f_n.

LEMMA 14. *If f is WRSR, then* $D^{G_f, 1/5} \to^{f_n} C^f$.

We work through the construction in reverse order. There will be four stages in this construction, the first three corresponding to the three levels of the definition of G and the last stage to the random self-reduction of f.

All stages are identical to those from the nonuniform proofs, but we need to verify that the use of nonuniformity (if any) can be replaced by an oracle for f_n. We'll just briefly review the constructions from other papers, to see that they are polytime computable from such an oracle, and refer to the relevant papers for proofs of correctness.

The four stages we need are given by the following lemmas, which together imply Lemma 14.

LEMMA 15 [27]. $D^{G_d, 1/5} \to^{f_n} C^{h, 1/2 + O(1/\ell)}$.

LEMMA 16 [13]. $C^{h, 1/2 + O(\ell^{-1})} \to C^{g, O(\ell^{-3})}$.

LEMMA 17 [22, 14, 20]. $C^{g, O(\ell^{-3})} \to^{f_n} C^{f, 1-n^{-c}}$.

Finally, by the definition of weak random self-reducibility $C^{f, 1-n^{-c}} \to C^f$.

Proof of Lemma 15. The construction is from [27]. Before starting, note that we can use an oracle for h_{n_2} given the oracle for f_n due to the above Turing reduction.

Let $D \in D_m^{G, 1/5}$. We construct a circuit to predict h as follows: Pick $i \in_U \{1, \ldots \ell\}$. For each $1 \leqslant j \leqslant \ell$ with $j \notin S_i$, pick $z_j \in_U \{0,1\}$. For each $i' < i$, query h at all $2^{|S_i \cap S_j|} \leqslant \ell$ strings that might be $z|_{S_{i'}}$ for a z consistent with the z_j's. Store the answered queries in a table T. Pick $b_{i'} \in_U \{0,1\}$ for $i \leqslant i' \leqslant \ell$.

Let C be the following circuit: on input x, set $z|_{S_i} = x$, while the other bits of z are fixed to the randomly chosen bits. Set $b_{i'} = h(z|_{S_{i'}})$ for $i' < i$, by looking up the appropriate entry in T. If $D(b_1, \ldots, b_\ell) = 1$ output b_i; else output $\neg b_i$.

By random sampling using the oracle for h_{n_2}, estimate the probability that $C(x) = h(x)$; if greater than $1/2 + 0.05/\ell$, output C, else repeat.

In [27], it is shown that the expected probability of success for C is at least $1/2 + .1/\ell$, so the number of repetitions before outputting a C that has good advantage is at most $O(n\ell)$ with very high probability. ▌

Proof of Lemma 16. Follows directly from [13]. Note that this part is completely uniform, and does not require an oracle. ▌

Proof of Lemma 17. This is the uniform version of a direct product lemma which has many proofs [22, 14, 20]. We present here the construction from [20] as being simple to describe.

Let $C \in C_{n_1}^{g,\gamma}$. Construct C' as follows: Let $n_3 = n_1/n = n^{c+1}$. Repeat for $r = 1$ to n^3/γ. Pick $i \in_U \{1, \ldots, n_3\}$. For each $j \neq i$, pick $x_j \in_U \{0, 1\}^n$, query $f(x_j)$ and record the answer. Flip coins until a head arises or until n tails have been flipped; let t_1 be the number of flips.

Let C'_r be the following three-valued circuit: On input x, compute t, the number of bit positions $j \neq i$ where the j'th bit of $C(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{n_3})$ disagrees with $f(x_j)$ (as recorded.) If $t < t_1$ output the ith bit of $C(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{n_3})$; otherwise output "reject."

Let C' be the circuit that outputs the majority answer from those C'_r that do not reject. In [20], it is proved that, for nonneglegible γ, $C' \in C^{f, 1-n^{-c}}$ with high probability. If γ is at least inverse polynomial, the construction takes polynomial time. ▌

2.5. The Derandomization

In the previous subsection we have used f_n to construct a sequence of generators $G_d^{f_n}$, each stretches a seed of n^c bits for some fixed constant c, into n^d bits for an arbitrarily specified d. We now show how to use these generators to derandomize *BPP* and *ZPP* algorithms in deterministic subexponential time 2^{k^δ} (when the input size is k), for an arbitrarily given constant $\delta > 0$.

Our main result in this section is that, if this simulation fails on *all* input length (the complement of succeeding infinitely often) for a given δ, then for some d, for all n, using an oracle to f_n, we can efficiently construct a distinguisher for $G_d^{f_n}$. Let f_n be f restricted to inputs of size n. For each input size n, we will construct a pseudorandom generator G_n from n^c bits for some fixed constant c, to n^d bits for an arbitrary $d > c$, that will be computable in polynomial time with an oracle for f_n. Given a circuit that distinguishes the output of this generator from truly random strings, we will be able to construct a circuit computing f on n bit strings, in polynomial time *with an oracle for* f_n.

The simulation of a *BPP* (resp. *ZPP*) algorithm (for any fixed language) is as follows. Let $\delta > 0$ be given. On inputs of size k, assume the *BPP* algorithm uses k^{c_1}

random bits and time. Set $d = 2cc_1/\delta$, and $n = k^{\delta/2c}$. Compute the range of G_n, a set of $n^d = k^{c_1}$ bit strings, in time $2O(n^c) = O(2^{k^\delta})$. Then simulate the *BPP* algorithm on each element of the range and take the majority vote.

Having defined the simulation, we are ready to formally state that if it fails, we have an efficient distinguisher.

LEMMA 18. *If* $BPP \not\subset i.o.-HeurTime_{1/n^c}(2^{n^\delta})$ *(resp. if* $ZPP \not\subset i.o.-AvgTime_{1/n^c}(2^{n^\delta})$*) for some* $c, \delta > 0$, *then* $D^{G_d^f, 1/5}$ *is efficiently constructible with oracle access to* f_n.

Proof. Assume that the above deterministic algorithm is incorrect with probability $1/k^d$ with respect to some sampleable distribution μ_k on k-bit strings, for all but finitely many k. Then given n, we can set $k = n^{2c/\delta}$ and sample instances $x_1, \ldots, x_r$ with $r = k^{O(1)}$ independently and randomly according to μ_k (by definition this can be done in probabilistic poly time in k and hence in n. With high probability the algorithm fails for one of the instances $x_1, \ldots, x_r$.

Let D_i be the circuit which views its input as a random sequence, and simulates the behavior of the *BPP* algorithm on x_i using that random sequence. This is a collection of circuits $D_1, \ldots, D_r$, produced in probabilistic polynomial time, so that, with high probability, at least one D_i distinguishes outputs of G_n from truly random strings.

So far, we don't know which of these circuits is a good distinguisher, but we have not used the oracle yet! We now use the fact that $G_d^{f_n}$ can be evaluated with oracle access to f_n, to test all of these D_i, and find one which is actually a distinguisher. Assuming that the error of the *BPP* algorithm was, say, $1/10$, the distinguisher we construct this was is in the set $D^{G_d^f, 1/5}$.

The proof for *ZPP* is identical. ∎

2.6. Removing the Oracle

Combining the main results of the previous two subsections, we see that if the conclusion of Theorem 5 fails, we have a probabilistic polytime algorithm which for every n, learns a circuit for f_n using an oracle for f_n, for every *WRSR* function f. However, we also know from [27, 7], that if the conclusion of Theorem 5 fails, then $EXP \in P/poly$. Combining this with the result of Karp and Lipton [21], we have $EXP \in \Sigma_2$. This, together with Toda's Theorem [30] and our Lemma 12 gives us an *EXP*-complete function f which is *DSR* (downward self-reducible), for which this learning algorithm exists. For a final contradiction, it remains to show that this learning algorithm can be turned (by removing the oracle access) into a *BPP* algorithm for f, contradicting the hypothesis of Theorem 5, which completes its proof.

LEMMA 19. *If* f *is DSR and* C^f *is efficiently constructible using oracle* f_n *then* $f \in BPP$.

Proof. We use downward self-reducibility in a manner reminiscent of the tester/corrector idea of Blum, Luby, and Rubinfeld ([9]). This will allow a efficient construction of C^f without oracle access, and this is equivalent to having a *BPP* algorith for the function.

We recursively compute circuits $C_1 \in C_1^f, \ldots, C_n \in C_n^f$. Say that we have computed C_i. We run the efficient construction algorith for C_{i+1}^f with oracle f_{i+1} (with error $\alpha = 1/n^2$), simulating queries to f_{i+1} by M^{C_i}, where M is the poly time oracle Turing Machine from the definition of downward self-reducibility.

Note that $|C_{i+1}| \leqslant$ (Time taken by the construction not counting oracle queries)* (Time taken to simulate queries *not counting the time to evaluate oracle calls by* M). This is a fixed polynomial in n, independent of the size of C_i. Since each $|C_i|$ is bounded by this fixed polynomial in n, the time for each stage (including time to evaluate oracle calls by M) is a fixed polynomial in n. Also, the probability that $C_n \notin C_n^f$ is at most $\alpha * n = 1/n$, so the error is bounded. ▌

3. PROOF OF THEOREM 6

Proof. We construct a problem in $E \cap P/poly$ that cannot be approximated in $TIME(2^{o(n)})/o(n)$ as follows. The idea is that some function from a universal family of hash functions will serve our purpose.

For any input of size n, we first simulate all machines with descriptions of size $.1n$ on all advice strings of size $.1n$ for 2^n steps on all inputs of length n. If they don't halt, record the answer as (say) 0. This gives us at most $2^{.2n}$ strings (truth tables) of $N = 2^n$ bits each. Let H be a family of $2^{O(n)}$ pairwise independent hash functions from n bits to 1 bit. (For example, $H = \{0, 1\}^n$, $h_r(x) = \langle r, x \rangle$). Consider them too as N-bit strings. Using Chebyshev bounds, one can see that a random $h_r \in H$ agrees with a string in $2/3N$ positions only with probability $1/O(N)$. Then since there are less than $O(N)$ strings in our collection, we can find a hash function that does not agree with any of them in $2/3$ the bits. We pick this h_r and output $h_r(x)$. ▌

4. CONCLUSIONS AND OPEN PROBLEMS

Ideally, we would hope that our results are a step towards proving $BPP \neq EXP$. However, our results provide reasons both to be optimistic and pessimistic about such a proof. On the one hand, our result makes such a result stronger, since it would show a positive simulation as well as a negative result. On the other, it clarifies that the best way of attacking this problem is to continue along the lines of the derandomization papers. It also shows that non-relativizing techniques can be useful in this area, so we need not be depressed by oracles where $BPP = EXP$. It also indicates that we do not need to prove circuit lower bounds to get such a result, so we should also be undaunted by the negative results on Natural Proofs ([28]).

There are some more technical points our work raises. First, is an average-case derandomization all one can hope for under a uniform assumption? If $BPP \neq EXP$ but $EXP \subset P/poly$, we have a paradoxical situation: the simulation of randomness by determinism does not always work, but it is intractable to find instances where it fails. Can we somehow utilize this intractibility in yet another layer of hardness vs. randomness trade-offs?

As can be seen, our main result is achieved essentially with no technical work. All that is taken from previous papers. On the other hand these papers are viewed from a somewhat different perspective in trying to make them uniform, which is subtle in some ways and raises some new questions regarding these issues.

The first one is that the classical "learning from a membership oracle" problem of computational learning theory arises naturally here. Let $LEARN$ be the class functions for which this can be done efficiently, namely all f for which C_n^f is constructible in PPT^{f_n}. (In the language of learning theory, $f \in LEARN$ if and only if the concept class consisting only of f_n is exactly learnable, using the hypotheses class of Boolean circuits, by membership queries in probabilistic polynomial-time.) This class is quite interesting, and we trivially have:

Fact 20. $BPP \subset LEARN \subset P/poly$.

Non-uniformity is essential; one can construct non-recursive elements of $LEARN$ by taking a parameterized family of concept classes exactly learnable through membership queries, and selecting one element of the class non-recursively for each n. $LEARN$ represents another way in which non-uniformity could help computation, in addition to randomness.

We showed that any downward self-reducible problem in $LEARN$ is also in BPP. What more can be said about the class $LEARN$?

Note that if we do not restrict the input size to the oracle queries (as is perhaps more natural in some learning settings, e.g., [5]), then it is probably not possible to eliminate the oracle.

Finally, we should mention that gaps similar to the one obtained here are possible at higher levels of the polynomial hierarchy, such as for whether $MA \neq NEXP$.

ACKNOWLEDGMENT

We are greatful to the referee for many valuable comments.

REFERENCES

1. A. Andreev, A. Clementi, and J. Rolim, Hitting sets derandomize *BPP*, *in* "XXIII International Colloquium on Algorithms, Logic and Programming (ICALP'96)," pp. 357–368, 1996.
2. A. Andreev, A. Clementi, and J. Rolim, Worst-case hardness suffices for derandomization: A new method for hardness–randomness tradeoffs, *Theor. Comp. Sci.* **221**(1–2) (1999), 3–18.
3. A. Andreev, A. Clementi, and J. Rolim, A new general derandomization method, *J. Assoc. Comput. Mach.* **45** (1998), 179–213.
4. A. Andreev, A. Clementi, J. Rolim, and L. Trevisan, Weak random sources, hitting sets, and *BPP* simulation, *SIAM J. Comput.* **28** (1999), 2103–2116.
5. N. Bshouty, R. Cleve, R. Gavalda, S. Kannan, and C. Tamon, Oracles and queries that are sufficient for exact learning, *J. Comput. System Sci.* **52** (1996), 421–433.
6. L. Babai, L. Fortnow, and C. Lund, Non-deterministic exponential time has two-prover interactive protocols, *Comput. Complexity* **1** (1991), 3–40.

7. L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP* has subexponential time simulations unless *EXPTIME* has publishable proofs, *Comput. Complexity* **3** (1993), 307–318.
8. D. Beaver and J. Feigenbaum, Hiding instance in multioracle queries, *in* "Proc. 7th Symposium on Theoretical Aspects of Computer Science," Lecture Notes on Computer Science, Vol. 415, pp. 37–48, 1990.
9. M. Blum, M. Luby, and R. Rubinfeld, Self-testing and self-correcting programs with applications to numerical programs, *Comput. Complexity* **3** (1993), 307–318.
10. M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* **13** (1984), 850–864.
11. J.-Y. Cai, A. Nerurkar, and D. Sivakumar, Hardness and hierarchy theorems for probabilistic time, *in* "31st Symposium on Theory of Computing," pp. 726–735, 1999.
12. O. Goldreich, H. Krawcyk, and M. Luby, On the existence of pseudorandom generators, *SIAM J. Comput.* **22** (1993), 1163–1175.
13. O. Goldreich and L. A. Levin, A hard-core predicate for all one-way functions, *in* "ACM Symp. on Theory of Computing," pp. 25–32, 1989.
14. O. Goldreich, N. Nisan, and A. Wigderson, On Yao's XOR-Lemma, ECCC TR 95–050, 1995.
15. J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby, A pseudorandom generator from any one-way function, *SIAM J. Comput.* **28** (1999), 1364–1396. [Preliminary versions by Impagliazzo *et al.* in "21st STOC," 1989, and Hastad in "22nd STOC," 1990.]
16. R. Impagliazzo, Hard-core distributions for somewhat hard problems, *in* "36th Foundations of Computer Science," pp. 538–545, 1995.
17. R. Impagliazzo, in preparation.
18. R. Impaglizzo, R. Shaltiel, and A. Wigderson, Near-optimal conversion of hardness into pseudo-randomness, *in* "40th Foundations of Computer Science," pp. 181–190, 1999.
19. R. Impaglizzo, R. Shaltiel, and A. Wigderson, Extractors and pseudo-random generators with optimal seed lengths, *in* "32nd Symposium on Theory of Computing," pp. 1–10, 2000.
20. R. Impagliazzo and A. Wigderson, $P = BPP$ unless E has sub-exponential circuits: Derandomizing the XOR Lemma, *in* "Proc. of the 29th Symposium on Theory of Computing," pp. 220–229, 1997.
21. R. M. Karp and R. J. Lipton, Turing machines that take advice, *L'Ensignment Math.* **28** (1982), 191–209.
22. L. A. Levin, One-way functions and pseudorandom generators, *Combinatorica* **7** (1987), 357–363.
23. M. Luby, "Pseudorandomness and Cryptographic Applications," Princeton Computer Science Notes, Princeton Univ. Press, Princeton, NJ, 1996.
24. L. A. Levin, Average case complete problems, *SIAM J. Comput.* **15** (1986) 285–286.
25. R. Lipton, New directions in testing, *in* "Distributed Computing and Cryptography" (J. Feigenbaum and M. Merritt, Eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 2, pp. 191–202, Amer. Math. Soc., Providence, RI, 1991.
26. N. Nisan, Pseudo-random bits for constant depth circuits, *Combinatorica* **11** (1991), 63–70.
27. N. Nisan and A. Wigderson, Hardness vs randomness, *J. Comput. System Sci.* **49** (1994), 149–167.
28. A. A. Razborov and S. Rudich, Natural proofs, *J. Comput. System Sci.* **55** (1997), 24–35.
29. A. Shamir, On the generation of cryptographically strong pseudo-random sequences, *in* "8th ICALP," Lecture Notes in Computer Science, Vol. 62, pp. 544–550, Springer-Verlag, Berlin, 1981.
30. S. Toda, *PP* is as hard as the polynomial–time hierarchy, *SIAM J. Comput.* **20** (1991), 865–877.
31. A. C. Yao, Theory and application of trapdoor functions, *in* "23rd Foundation of Computer Science," pp. 80–91, 1982.