Derandomization: A Brief Overview*

Valentine Kabanets†
School of Computing Science
Simon Fraser University
Vancouver, Canada
kabanets@cs.sfu.ca

September 6, 2003

Abstract

This survey focuses on the recent (1998–2003) developments in the area of derandomization, with the emphasis on the derandomization of time-bounded randomized complexity classes.

1 Introduction

1.1 History

The last twenty years have witnessed the abundance of efficient randomized algorithms developed for a variety of problems [MR95]. The class BPP has effectively superseded the class P as the class of problems that are considered efficiently solvable, while many researchers believe that BPP = P.

There are, essentially, two general arguments to support the belief that BPP is “close” to P. The first argument is empirical: a large number of randomized algorithms have been implemented and seem to work just fine, even without access to any source of true randomness. The second argument is that every language in BPP can be nontrivially derandomized, i.e., decided deterministically in subexponential time, if certain combinatorial objects of “high” nonuniform complexity can be “efficiently” uniformly constructed.

1.1.1 Hardness-randomness tradeoffs

The second argument makes use of so-called hardness-randomness tradeoffs, the results showing that “computational hardness” can be efficiently converted into “computational randomness”. The possibility of trading hardness for randomness was first suggested in [Sha81, BM84, Yao82]. Yao [Yao82, BH89] demonstrated that a one-way permutation, a permutation which is “easy” to compute but “hard” to invert on the average, can be used to construct a pseudorandom generator; later, the assumption has been weakened to say that any one-way function would suffice [Lev87, GKL88, GL89, HILL99]. Informally, a pseudorandom generator is an efficiently computable function mapping “short” input strings to “longer” output strings so that the uniform distribution on the inputs to the generator induces the distribution on the outputs that “looks”
uniform to any computationally restricted observer. Such a pseudorandom generator can be used to simulate any BPP algorithm $A$ in deterministic subexponential time as follows: for each input to the generator, compute the output $w$ of the generator, and compute the decision of the algorithm $A$ using $w$ as a string of random bits; output the majority decision.

1.1.2 The Nisan-Wigderson generator

Observe that the running time of the deterministic simulation of a given BPP algorithm using a pseudorandom generator is dominated by the amount of time needed to enumerate all inputs to the generator. Thus, if the goal is to derandomize BPP, then it makes sense to relax the efficiency requirements in the definition of a pseudorandom generator by allowing the generator to run in time exponential in the input size.

Nisan and Wigderson [NW94], based on [Nis91], showed how to make use of this observation by constructing an exponential-time computable hardness-based generator, the NW generator. This generator is pseudorandom if the class $\text{EXP} = \text{DTIME}(2^{\text{poly}(n)})$ contains a language $L$ hard on the average with respect to “small” Boolean circuits, i.e., no small circuit can correctly decide the language $L$ on significantly more than a half of all the inputs of any given length.

The conditional derandomization of BPP obtained in [NW94] was an improvement on Yao’s result, since the existence of a one-way function can be shown to imply the existence of a language $L$ in $\text{EXP}$ such that $L$ cannot be well approximated by any family of small circuits. It was natural, however, to try to strengthen the Nisan-Wigderson tradeoffs by replacing the assumption of average-case hardness with that of worst-case hardness.

1.1.3 Worst-case hardness-randomness tradeoffs

The first worst-case hardness-randomness tradeoff was achieved by Babai, Fortnow, Nisan, and Wigderson [BFNW93]. They showed that if $\text{EXP}$ contains a language of superpolynomial circuit complexity (i.e., $\text{EXP} \not\subset \text{P}/\text{poly}$), then every BPP algorithm can be simulated deterministically in subexponential time, for infinitely many input lengths. The main new idea in [BFNW93], inspired by the results on the random self-reducibility of low-degree polynomials [BF90, Lip91] and the work on two-prover interactive protocols for $\text{NEXP}$ [BFL91], was the use of error-correcting codes to convert the truth table of a Boolean function hard in the worst case into that of a Boolean function hard on the average.

The methods of [BFNW93], however, failed to show that the conclusion $\text{BPP} = \text{P}$ can be derived from some worst-case hardness assumption; the average-case version of such a tradeoff was known from [NW94]. The encoding used in [BFNW93], the extension of a Boolean function to a low-degree multivariate polynomial over a finite field, enabled one to obtain a Boolean function that is only “mildly” hard on the average from the Boolean function that is “very” hard in the worst case — or so it seemed at the time. Yao’s XOR Lemma [Yao82, GNW95] could be used to amplify the average-case hardness of a given Boolean function $f$ by XORing the values $f(x_i)$ for a few independent inputs $x_i$ to $f$. But, to attain the level of average-case hardness sufficient for concluding $\text{BPP} = \text{P}$, one would need to use too many independent bits.

Impagliazzo and Wigderson [IW97], using [Imp95], showed that Yao’s XOR Lemma can be derandomized in a rather dramatic sense: it remains true even if $n^{O(1)}$ inputs $x_i$ to $f$ are constructed using only $O(n)$ independent bits. Combined with the previous results, this yields the desired “high-end” worst-case hardness-randomness tradeoff: if $\text{E} = \text{DTIME}(2^{O(n)})$ contains a language of circuit complexity $2^{\Omega(n)}$ almost everywhere, then $\text{BPP} = \text{P}$. 

2
1.1.4 Hitting-set generators

Independently of [IW97], and using different methods, Andreev, Clementi, and Rolim [ACR97], based on [ACR98], showed that $\text{BPP} = \text{P}$ if there is an efficiently enumerable sparse language of high circuit complexity; here “high” means that the circuit complexity of deciding the language is close to that of generating the language. Rather than constructing a pseudorandom generator, the authors of [ACR97] showed that their hardness assumption implies the existence of a hitting-set generator, the generator whose output distribution “looks” random to any $\text{RP}$ algorithm, rather than $\text{BPP}$ algorithm. Somewhat surprisingly, the existence of an efficient hitting-set generator implies that $\text{BPP} = \text{P}$ [ACR98] (see also [ACRT99, BF99, GW99, GVW00]).

1.2 Recent developments

From 1998, the research on derandomization can be roughly divided into the following categories:

1. improvements of the hardness-randomness tradeoffs,
2. applications of the hardness-randomness tradeoffs in more general complexity-theoretic and information-theoretic settings,
3. “uniform” hardness-randomness tradeoffs,
4. limitations of the current derandomization techniques.

These developments will be addressed in the rest of this survey — albeit at a highly intuitive level. For more information on pseudorandomness and derandomization, the readers are referred to the excellent presentations by Goldreich [Gol99, Chapter 3] and Miltersen [Mil01].

2 Better Tradeoffs

2.1 Hardness amplification via error-correcting codes

Impagliazzo and Wigderson [IW97] showed how to obtain a Boolean function of high average-case hardness, starting with a Boolean function of high worst-case hardness. The first step was the low-degree polynomial encoding of the truth table of a given worst-case hard Boolean function, as in [BFNW93]. This produced a Boolean function that is “mildly” hard on the average: any small circuit can compute the new function correctly on at most $1 - \frac{1}{\text{poly}(n)}$ fraction of all $n$-bit inputs. The second step was the amplification of the average-case hardness via the derandomized version of Yao’s XOR Lemma, so that no small circuit can compute the resulting function on more than $\frac{1}{2} + \frac{1}{2\text{poly}(n)}$ fraction of all $n$-bit inputs.

Sudan, Trevisan, and Vadhan [STV01] show that the first step alone, with a different choice of parameters, already gives the desired high average-case hardness, and so no further hardness amplification is needed. As explained in [STV01], there is an intimate connection between the “worst-case to average-case” reductions and the task of list-decoding of error-correcting codes, which will be sketched below.

Given the truth table of an $n$-variable Boolean function $f$, of length $N = 2^n$, that has circuit complexity $2^{\Omega(n)}$, the task of a “worst-case to average-case” reduction is to produce the truth table of a new Boolean function $g$ on $O(n)$ variables that any circuit of size at most $2^{\Omega(n)}$ can compute on at most $\frac{1}{2} + \frac{1}{2\text{poly}(n)}$ fraction of all inputs; this reduction must be computable by a uniform algorithm, which justifies the use of order notation. To argue that $g$ has the required hardness, one needs to
show how a small circuit computing \( g \) on at least \( \frac{1}{2} + \frac{1}{2m} \) fraction of inputs gives rise to a small circuit computing \( f \) everywhere, thus contradicting the worst-case hardness of \( f \).

On the other hand, the goal of an error-correcting encoding is to map, via the encoding function \( \text{Enc} \), a given message string \( m \) of length \( N \) into a codeword string \( c = \text{Enc}(m) \) of bigger length so that the original message \( m \) can be efficiently recovered, via the decoding function \( \text{Dec} \), from any string \( r \) that is sufficiently close to \( c \) in the Hamming distance.

To see the connection with the “worst-case to average-case” reduction, we should think of the message \( m \) as the truth table of a Boolean function \( f \), the codeword \( c = \text{Enc}(m) \) as the truth table of a new Boolean function \( f' \), and the string \( r \) as the truth table of a Boolean function computed by some small circuit.

In order to get the desired parameters in the “worst-case to average-case” reduction, we would need the binary error-correcting codes, given by the encoding and decoding functions \( \text{Enc} \) and \( \text{Dec} \), with the following properties:

1. \( |\text{Enc}(m)| \in \text{poly}(|m|) \),
2. the message \( m \) can be recovered from a string \( r \) whenever \( r \) and \( c = \text{Enc}(m) \) agree in at least \( \frac{1}{2} + \frac{1}{|c|^{1/2}} \) fraction of positions.

However, by the Plotkin bound [Plo60], it is impossible to achieve both these conditions for any binary code, if one insists that decoding be unique. The solution is to allow the decoding procedure to return a short list containing all the codewords \( c \) that are sufficiently close to the received word \( r \), i.e., to use list-decodable error-correcting codes (see [Sud00] for a survey on list decoding).

Another important issue is the efficiency of the decoding procedure. In the standard setting, a decoding procedure is considered efficient if it runs in time polynomial in the length of the received string. This is not good enough in our case. We need the decoding procedure to compute each individual bit in the recovered codeword \( c \) in sublinear (even polylogarithmic) time in order to claim the existence of a small circuit for the Boolean function whose truth table is \( c \). So, for our purposes, an efficient decoding procedure is the one that, given oracle access to the received string \( r \), outputs a short list of small oracle circuits such that every codeword close to \( r \) will be computed by some circuit on the list, given oracle access to \( r \); here, oracle access to \( r \) means that each individual bit of \( r \) can be looked up in constant time.

Amazingly, the binary codes satisfying the requirements stated above do exist, and can be constructed using low-degree multivariate polynomials. The existence of an efficient list-decoding algorithm for these codes follows from the work of Arora and Sudan [AS97], using [Sud97]; a simpler algorithm with improved parameters is given in [STV01]. The nice properties of these list-decoding algorithms are based, in particular, on the existence of efficient algorithms for interpolating and factoring polynomials.

### 2.2 Achieving optimal hardness-randomness tradeoffs

To discuss what it means to have an optimal hardness-randomness tradeoff, we need to recall some definitions. Below, by a function, we will actually mean a family of functions, parameterized by the input size. Assuming that a circuit need not use all of its inputs, we may talk about circuits of size \( n \) on \( n \) inputs.

A function \( G : \{0,1\}^n \rightarrow \{0,1\}^m \), where \( m = m(n) \) is some function of \( n \), is called a pseudo-random generator (PRG) if, for any Boolean circuit \( C \) of size \( m \) on \( n \) inputs,

\[
|\Pr_x[C(x) = 1] - \Pr_y[C(G(y)) = 1]| < \frac{1}{m},
\]
where $x$ and $y$ are chosen uniformly at random from $\{0,1\}^m$ and $\{0,1\}^n$, respectively. The PRG $G$ is said to produce $m$ bits of pseudorandomness using a seed of size $n$. The PRG computable in time $2^{O(n)}$ is called quick.

A Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ has hardness $s$, where $s = s(n)$ is some function of $n$, if the size of the smallest Boolean circuit computing $f$ is at least $s$.

The following observation has appeared in several papers (e.g., [ISW99]).

**Theorem 1.** If there is a quick PRG $G : \{0,1\}^n \rightarrow \{0,1\}^m$, then there is a $2^{O(n)}$-time computable Boolean function $f : \{0,1\}^{n+1} \rightarrow \{0,1\}$ of hardness $m$.

**Proof.** Consider the Boolean function $f$ defined as follows: for every $x \in \{0,1\}^{n+1}$,

$$f(x) = 0 \Leftrightarrow x \in \{G(y)_{1..n+1} \mid y \in \{0,1\}^n\},$$

where $G(y)_{1..k}$ denotes the $k$-length prefix of the string $G(y)$.

It is easy to see that $f$ is $2^{O(n)}$-time computable. Also note that $p = \Pr_x[f(x) = 1] \geq \frac{1}{2}$, since $f(x) = 0$ for at most $2^n$ outputs of the generator.

Suppose that the function $f$ is computable by a Boolean circuit of size at most $m$. Then the PRG $G$ can be used to approximate the value $p$, to within $1/m$. However, $f$ is defined so that

$$\Pr_y[f(G(y)) = 1] = 0 < p - 1/m,$$

for any $m > 2$. Thus, $f$ must have hardness at least $m$. \hfill \Box

Theorem 1 essentially says that $m$ bits of pseudorandomness, using a seed of size $n$, yield a $O(n)$-input Boolean function with hardness $m$. The tradeoffs of [BFNW93, IW97] prove the converse to Theorem 1 for the specific values of the parameter $s(n)$, the hardness of an $n$-input Boolean function. In particular, [IW97] proves that hardness $s(n) = 2^{\Omega(n)}$ yields $m = (s(\log n))^{\Omega(1)} = n^{\Omega(1)}$ bits of pseudorandomness via a PRG from $\log n$ to $m$ bits.

In general, the results showing that a $2^{O(n)}$-time computable $n$-input Boolean function with hardness $s = s(n)$ yields $m = s^{\Omega(1)}$ bits of pseudorandomness via a quick PRG from $O(n)$ to $m$ bits are considered optimal hardness-randomness tradeoffs, up to a polynomial. Almost optimal tradeoffs were established in [ISW99, ISW00]; they were based upon a recursive use of the NW generator.

Building upon the techniques from [STV01, MV99, TSZS01], Shaltiel and Umans [SU01] prove an optimal hardness-randomness tradeoff for hitting-set generators, rather than PRGs. Combined with the methods from [ACR98, GVV00], this implies the following optimal derandomization of BPP, assuming the existence of hard functions.

**Theorem 2 ([SU01]).** If there is a $2^{O(n)}$-time computable Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ of hardness $s = s(n)$, then $BPTIME(t) \subseteq DTIME(2^{O(s^{-1}(t^{O(1)}))})$, where $t = t(n)$ is a function of $n$.

An interesting aspect of the methods in [SU01] is that they show how to convert worst-case hardness into pseudorandomness without applying the NW generator; the previous constructions of PRGs relied upon the NW generator as a method to convert average-case hardness into pseudorandomness.

The techniques in [SU01] make essential use of the error-correcting properties of polynomial codes and the algebraic structure of vector spaces over finite fields. Extending these techniques, Umans [Uma02] obtains the following optimal hardness-randomness tradeoff for PRGs, which also implies Theorem 2.

**Theorem 3 ([Uma02]).** If there is a $2^{O(n)}$-time computable $n$-input Boolean function of hardness $s = s(n)$, then there is a quick PRG from $O(n)$ to $s^{\Omega(1)}$ bits.
3 Diverse Applications of the Hardness-Randomness Tradeoffs

3.1 Beyond BPP

Originally, the hardness-randomness tradeoffs were motivated by the task of derandomizing such probabilistic complexity classes as BPP and RP. Following Yao [Yao82], the goal was to construct a suitable pseudorandom generator that can be used to approximate the acceptance probability of any given small Boolean circuit. But, the existence of such pseudorandom generators would imply much more than the derandomization of BPP.

As shown by Goldreich and Zuckerman [GZ97], one fairly straightforward implication is the derandomization of the class MA defined by Babai [Bab85, BM88]. Recall that a language $L \in \text{MA}$ if there is a polynomial-time computable relation $R_L$ such that, for any string $x$,

$$x \in L \Rightarrow \exists y : \Pr_z [R_L(x, y, z) = 1] > \frac{3}{4},$$

$$x \notin L \Rightarrow \forall y : \Pr_z [R_L(x, y, z) = 1] < \frac{1}{4},$$

where $|y| = |z| = |x|^{O(1)}$.

Since $R_L$ is polynomial-time computable, it is also computable by a family of polynomial-sized Boolean circuits. The existence of a quick PRG, say from $O(\log n)$ to $n$ bits, would allow us to estimate the probability $\Pr_z [R_L(x, y, z) = 1]$ deterministically in polynomial time, and hence imply that $\text{MA} \subseteq \text{NP}$. Thus, the known hardness-randomness tradeoffs show that the existence of a language in $\text{E}$ of high circuit complexity implies the derandomization of $\text{MA}$.

The situation with the class AM [Bab85, BM88], which contains MA, is trickier. By definition, a language $L \in \text{AM}$ if there is a polynomial-time computable relation $R_L$ such that, for every string $x$,

$$x \in L \Rightarrow \exists y : R_L(x, y, z) = 1 \geq \frac{3}{4},$$

$$x \notin L \Rightarrow \forall y : R_L(x, y, z) = 1 < \frac{1}{4},$$

where $|y| = |z| = |x|^{O(1)}$.

To derandomize AM, we would need to estimate the acceptance probability of a non-deterministic Boolean circuit deciding, for given $x$ and $z$, whether there is a $y$ such that $R_L(x, y, z) = 1$. Thus, the existence of a PRG does not seem to suffice.

Klivans and van Melkebeek [KM99] point out that the Boolean function $f(x, z) = 1 \iff \exists y : R_L(x, y, z) = 1$ is in $\text{P}^{\text{NP}}$, and thus is computable by a family of polynomial-sized Boolean circuits with oracle access to SAT. So, the existence of a PRG that estimates the acceptance probability of any small SAT-oracle Boolean circuit would imply the derandomization of AM.

The crucial observation in [KM99] is that all known hardness-randomness tradeoffs relativize. In particular, for any oracle $A$, the truth table of a Boolean function of high $A$-oracle circuit complexity gives rise to a PRG whose output distribution “looks random” to any small Boolean circuit with $A$-oracle gates. The relativized hardness-randomness tradeoffs yield, e.g., the following result; recall that $\text{NE} = \text{NTIME}(2^{O(n)})$.

**Theorem 4 ([KM99]).** If $\text{NE} \cap \text{coNE}$ contains a language of SAT-oracle circuit complexity $2^{\Omega(n)}$ almost everywhere, then $\text{AM} = \text{NP}$.

Miltersen and Vinodchandran [MV99] improve upon Theorem 4 by replacing the assumption of high SAT-oracle circuit complexity with that of high non-deterministic circuit complexity; the average-case version of such a tradeoff was proved earlier in [AK97]. The methods in [MV99]
build upon those from [ACR98, ACR97] for constructing hitting-set generators; an important new ingredient in [MV99] is the use of certain polynomial error-correcting codes. Further improvements are obtained in [SU01, Uma02].

Klivans and van Melkebeek [KM99] apply the relativized hardness-randomness tradeoffs to get conditional derandomization of a number of probabilistic constructions. In particular, they derandomize the Valiant-Vazirani random hashing algorithm [VV86].

**Theorem 5 ([KM99]).** If $E$ contains a language of SAT-oracle circuit complexity $2^{O(n)}$ almost everywhere, then the following task can be performed deterministically in polynomial time: given a propositional formula $\phi$, generate a list of propositional formulas such that

- if $\phi$ is unsatisfiable, then so is every formula on the list, and
- if $\phi$ is satisfiable, then at least one of the formulas on the list has exactly one satisfying assignment.

The proof is based on the fact that there is a $P^{NP}$ algorithm for checking if a given propositional formula has exactly one satisfying assignment. Hence, it suffices to build a PRG whose output distribution “looks random” to any polynomial-size SAT-oracle circuit.

### 3.2 Beyond computational complexity

Viewed abstractly, a hardness-randomness tradeoff is an efficient transformation of a binary string $x$, the truth table of a Boolean function on $\log|x|$ inputs, to the distribution $D_x$ on binary strings $y$, where $y$’s are the outputs of the PRG based on $x$, such that the following holds: any statistical test $T(y)$ distinguishing the distribution $D_x$ from the uniform distribution can be used, together with some “short” advice string $a$ dependent on $x$, as a description of the string $x$.

In the applications of hardness-randomness tradeoffs to derandomizing BPP or AM, the statistical tests $T(y)$ are Boolean functions computable by small circuits or SAT-oracle circuits. The idea is that if the acceptance probability of a circuit $C$ is not approximated correctly by the given PRG based on a Boolean function $f$, then $C$ can be used to construct a “small” circuit computing $f$; this leads to a contradiction if $f$ is of high circuit complexity.

Trevisan [Tre99] demonstrated the usefulness of hardness-randomness tradeoffs in the information-theoretic setting, where the statistical test $T(y)$ can be an arbitrary Boolean function, not necessarily computable by a small circuit. The reasoning is, roughly, as follows.

Let $S \subseteq \{0,1\}^n$ be any set. Let $T_0 : \{0,1\}^k \rightarrow \{0,1\}$ be an arbitrary Boolean function, possibly dependent on $S$. Define $S_0 \subseteq S$ to be the subset of all those strings $x_0$ such that $T_0$ distinguishes the distribution $D_{x_0}$ from uniform, where $D_x$ is a distribution on $k$-bit strings. Then every string $x_0 \in S_0$ is uniquely determined by $T_0$ together with some short advice string $a$ (dependent on $x_0$), where $|a| \ll n$. Since there are few short strings, the set $S_0$ must be small.

Now, consider the distribution $E_S$ on $k$-bit strings defined as follows. Choose $x \in S$ uniformly at random, and output a string $y$ sampled according to the distribution $D_x$. The distribution $E_S$ must be statistically close to uniform.

Indeed, suppose that $E_S$ is far from uniform. Then there is a statistical test $T_0 : \{0,1\}^k \rightarrow \{0,1\}$ distinguishing this distribution from uniform. By a Markov-style argument, there must be a large subset $S_0 \subseteq S$ such that, for every $x \in S_0$, the test $T_0$ distinguishes $D_x$ from uniform. But this is impossible since, by the discussion given above, $S_0$ should be small.

This reasoning led Trevisan [Tre99] to a breakthrough in the construction of extractors, efficiently computable functions $E(x, s)$ that can be used to convert a source of “weak” randomness
into a source of statistically “almost” uniform randomness, using a short truly random seed. The distribution $E_S$ described above is an example of an extractor, where the set $S$ is used as a source of weak randomness and the additional truly random short seed $s$ is used to sample from $D_x$.

The connection between PRGs and extractors, discovered in [Tre99], has played an important role in many recent results on extractors; see [Sha02] for a survey.

3.3 Back to computational complexity

Trevisan [Tre99] showed that the proof technique originally used for constructing PRGs can also be very useful in constructing extractors. The correctness proof of such extractor constructions relies upon a “decoding” procedure for strings $x$ sampled from a source of weak randomness. Let $E_x$ be the distribution induced by an extractor $E(x, s)$ when $x$ is fixed. Then, given a statistical test distinguishing the distribution $E_x$ from uniform and a short advice string $a$, this “decoding” procedure must uniquely determine the string $x$.

The natural question is whether such an extractor construction should yield a PRG construction. After all, the correctness proofs in both cases rely upon certain “decoding” procedures. The important difference, however, is the efficiency requirement: the efficiency of “decoding” is not important in the setting of extractors, but it is crucial in the setting of PRGs.

Nonetheless, the connection between PRGs and extractors has been exploited in the opposite direction! Shaltiel and Umans [SU01, Uma02] start with the extractor proposed by Ta-Shma, Zuckerman, and Safra [TSZS01] and, employing a lot of new ideas, show how to turn it into a PRG. Moreover, the resulting PRG gives an optimal hardness-randomness tradeoff (see Section 2.2).

4 Towards Uniform Hardness-Randomness Tradeoffs

4.1 Derandomizing BPP

The hardness-randomness tradeoffs considered so far show that a language in $\text{EXP}$ of high nonuniform (i.e., circuit) complexity yields a quick generator that is pseudorandom with respect to any nonuniform family of small circuits. That is, a nonuniform hardness assumption yields a PRG for nonuniform algorithms.

Intuitively, it is reasonable to conjecture that a uniform hardness assumption should yield a PRG for uniform algorithms. In particular, one might conjecture that $\text{EXP} \not\subseteq \text{P}$ should yield a PRG for any $\text{P}$-uniform family of polynomial-size Boolean circuits. Unfortunately, the existence of such a PRG has not been proved yet.

However, Impagliazzo and Wigderson [IW98] prove the following version of a uniform hardness-randomness tradeoff.

**Theorem 6 ([IW98]).** If $\text{EXP} \not\subseteq \text{BPP}$, then, for every $\epsilon > 0$, there is a quick generator $G : \{0,1\}^{n^\epsilon} \rightarrow \{0,1\}^n$ that is pseudorandom with respect to any $\text{P}$-sampleable family of $n$-size Boolean circuits infinitely often.

The phrase “$G$ is pseudorandom with respect to any $\text{P}$-sampleable family of circuits infinitely often” means the following. Let $B_G(n)$ be the set of all Boolean circuits $C$ of size $n$ that are “bad” for the generator $G$, i.e., $C \in B_G(n)$ iff

$$|\Pr_x[C(x) = 1] - \Pr_y[C(G(y)) = 1]| \geq 1/n.$$
Let $R$ be any probabilistic polynomial-time algorithm that, on input $1^n$, outputs a Boolean circuit of size $n$. Then there are infinitely many $n$ such that
\[ \Pr[R(1^n) \in B_G(n)] < \frac{1}{n}, \]
where the probability is over the internal coin tosses of $R$.

**Proof Sketch of Theorem 6.** If $\text{EXP} \not\subset \text{P/poly}$, then Theorem 6 follows by the standard (nonuniform) hardness-randomness tradeoff from [BFNW93]. On the other hand, if $\text{EXP} \subset \text{P/poly}$, then $\text{EXP}$ collapses to $\Sigma^p_2$ [KL82], and since $\Sigma^p_2 \subseteq \#P$ [Tod91], we conclude that $\#P$-complete languages are also complete for $\text{EXP}$. Thus, it suffices to consider a generator based on $\text{PERMANENT}$ [Val79].

Inspecting the correctness proof of the hardness-randomness tradeoff in [BFNW93] reveals the following. If the $\text{PERMANENT}$-based generator can be broken by a $\text{BPP}$ algorithm, then a polynomial-size circuit computing $\text{PERMANENT}_n$ (on $n$-bit inputs) can be learned in probabilistic polynomial time, given oracle access to $\text{PERMANENT}_n$; the existence of this learning algorithm depends on the random self-reducibility of $\text{PERMANENT}$.

The fact that $\text{PERMANENT}$ is also downward self-reducible can then be exploited to remove the need for an oracle. Namely, to construct a circuit $C_n$ computing $\text{PERMANENT}_n$, we first construct small circuits $C_1, \ldots, C_{n-1}$ computing $\text{PERMANENT}_1, \ldots, \text{PERMANENT}_{n-1}$, respectively. Then we run the probabilistic learning algorithm to construct $C_n$, using the previously constructed circuit $C_{n-1}$ to answer any oracle queries about $\text{PERMANENT}_n$. This shows that $\text{PERMANENT}$ is in $\text{BPP}$, and hence, $\text{EXP} = \text{BPP}$.

An immediate corollary of Theorem 6 is the “uniform” derandomization of $\text{BPP}$ under the assumption that $\text{EXP} \neq \text{BPP}$.

**Theorem 7 ([IW98]).** If $\text{EXP} \neq \text{BPP}$, then, for any $\epsilon > 0$, every $\text{BPP}$ algorithm can be simulated deterministically in time $2^{n^\epsilon}$ so that, for infinitely many $n$, this simulation is correct on at least $1 - \frac{1}{n}$ fraction of all inputs of size $n$.

Unlike the proofs of standard (nonuniform) hardness-randomness tradeoffs, the proof of Theorem 6 relies upon nonrelativizing techniques; in particular, the proof uses the nonrelativizing result from [KL82] saying that $\text{EXP} \subset \text{P/poly} \Rightarrow \text{EXP} = \Sigma^p_2$. It is not known, however, whether Theorem 6 itself relativizes.

Trevisan and Vadhan [TV02] give a different proof of Theorem 6; their proof does *not* rely upon the theorems of Toda [Tod91] and Valiant [Val79], but rather is based on the ideas from the proof of $\text{IP} = \text{PSPACE}$ [LFKN92, Sha92]. Another result in [TV02] is an optimal “worst-case to average-case” reduction for $\text{EXP}$ in the uniform setting, with the parameters matching those in the nonuniform setting [STV01].

### 4.2 Derandomizing $\text{RP}$

It is possible to prove a version of Theorem 6 using the weaker assumption $\text{EXP} \neq \text{ZPP}$. We need to modify our setting.

For a generator $H : \{0, 1\}^k \rightarrow \{0, 1\}^n$, let $B_H(n)$ be the set of all circuits $C$ of size $n$ such that $\Pr_x[C(x) = 1] \geq 1/2$ but $\Pr_y[C(H(y)) = 1] = 0$; that is, the circuits in $B_H(n)$ show that $H$ is not a hitting-set generator.
The generator $H$ is called a hitting-set generator with respect to any $P$-sampleable family of $n$-size Boolean circuits infinitely often if the following holds. For any probabilistic polynomial-time algorithm $R$, where $R(1^n)$ outputs a Boolean circuit $C$ of size $n$, there are infinitely many $n$ where

$$\Pr[R(1^n) \in B_H(n)] < 1.$$ 

**Theorem 8 ([Kab01]).** If $\text{EXP} \not\subseteq \text{ZPP}$, then, for every $\epsilon > 0$, there is a quick generator $H : \{0,1\}^n \rightarrow \{0,1\}^n$ that is a hitting-set generator with respect to any $P$-sampleable family of $n$-size Boolean circuits infinitely often.

The proof of Theorem 8 uses the “easy witness” generator $\text{Easy} : \{0,1\}^k \rightarrow \{0,1\}^n$ defined as follows. For any $y \in \{0,1\}^k$, $\text{Easy}(y) = t$ where $t$ is the truth table of a log $n$-input Boolean function computed by the Boolean circuit described by the string $y$.

**Proof Sketch of Theorem 8.** The main idea is that if $\text{Easy} : \{0,1\}^n \rightarrow \{0,1\}^n$ can be uniformly broken for some $\epsilon > 0$, then $\text{BPP} = \text{ZPP}$.

Indeed, suppose that the generator $\text{Easy}$ is not a hitting-set generator with respect to some $P$-sampleable family of $n$-size Boolean circuits, almost everywhere. This means that, for all sufficiently large $n$, we can efficiently generate some Boolean circuit $C$ of size $n$ such that (i) $C$ accepts at least $1/2$ of all $n$-bit strings and (ii) every $n$-bit string accepted by $C$ has circuit complexity greater than $n^\epsilon$. Consequently, we can probabilistically guess, with zero error, a hard string and convert it into pseudorandomness via the known hardness-randomness tradeoffs. The conclusion $\text{BPP} \subseteq \text{ZPP}$ follows.

Thus, if the generator $\text{Easy}$ does not work, then $\text{BPP} = \text{ZPP}$. On the other hand, if the conclusion of Theorem 8 is false, then so is the conclusion of theorem 6, and hence $\text{EXP} = \text{BPP}$. 

A corollary of Theorem 8 is the following unconditional result about the “easiness” of $\text{RP}$ in a certain uniform setting.

**Theorem 9 ([Kab01]).** At least one of the following holds.

1. $\text{RP} \subseteq \text{ZPP}$.

2. For any $\epsilon > 0$, every $\text{RP}$ algorithm can be simulated in deterministic time $2^{n^\epsilon}$ so that, for any polynomial-time computable function $f : \{1\}^n \rightarrow \{0,1\}^n$, there are infinitely many $n$ where this simulation is correct on the input $f(1^n)$.

### 4.3 Derandomizing AM

Lu [Lu00] considers the modified generator $\text{Easy}_{\text{SAT}} : \{0,1\}^k \rightarrow \{0,1\}^n$ that, on input $y$, outputs the truth table of the Boolean function computable by a SAT-oracle circuit whose description is $y$. If this modified generator can be uniformly broken almost everywhere, then we can guess, with zero error, a Boolean function of high SAT-oracle circuit complexity. Plugging this function into the known hardness-randomness tradeoffs, we can derandomize AM (see Theorem 4).

Using $\text{Easy}_{\text{SAT}}$ to search for $\text{NP}$-witnesses, i.e., checking if any output of $\text{Easy}_{\text{SAT}}$ is a satisfying assignment for a given propositional formula, Lu obtains the following.

**Theorem 10 ([Lu00]).** At least one of the following holds.

1. $\text{AM} \subseteq \text{NP}$.
2. For any $\epsilon > 0$, every NP (and every coNP) algorithm can be simulated in deterministic time $2^n\epsilon$ so that, for any polynomial-time computable function $f : \{1\}^n \to \{0, 1\}^n$, there are infinitely many $n$ where this simulation is correct on the input $f(1^n)$.

Since the Graph Nonisomorphism Problem (GNI) belongs to both AM [GMW91, GS89, BM88] and coNP, Theorem 10 implies that either GNI is in NP or GNI can be simulated in deterministic subexponential time so that this simulation appears correct with respect to any deterministic polynomial-time computable function $f : \{1\}^n \to \{0, 1\}^n$.

More recently, Gutfreund, Shaltiel, and Ta-Shma [GSTS03] proved a version of Theorems 6 and 8 for the class AM.

**Theorem 11 ([GSTS03]).** If $\mathbf{E} \not\subseteq \text{AM-TIME}(2^{\epsilon n})$ for some $\epsilon > 0$, then every language $L \in \text{AM}$ has an NP algorithm $A$ such that, for every polynomial-time computable function $f : \{1\}^n \to \{0, 1\}^n$, there are infinitely many $n$ where the algorithm $A$ correctly decides $L$ on the input $f(1^n)$.

Like Theorems 6 and 8, Theorem 11 can be interpreted as a “gap” theorem. Informally, it says that either AM is almost as powerful as E, or AM is no more powerful than NP from the point of view of any efficient observer.

The proof of Theorem 11 relies on certain special properties of a hitting-set generator for AM constructed in [MV99]. This generator allows one to obtain only a “high-end” tradeoff: if $\mathbf{E}$ requires exponential time to be decided by an AM protocol, then AM is “close” to NP. It is an interesting open question whether a “low-end” tradeoff for AM is also true: if $\mathbf{EXP} \not\subseteq \text{AM}$, then AM can be simulated in nondeterministic subexponential time so that the simulation looks correct to any efficient observer, infinitely often.

## 5 Hitting the Wall?

### 5.1 Circuit lower bounds from the derandomization of MA

Hardness-randomness tradeoffs have been hailed as a step forward in the quest to prove that $\mathbf{BPP} = \mathbf{P}$: once superpolynomial circuit lower bounds are proved for some language in EXP, the derandomization of BPP will follow. However, proving superpolynomial circuit lower bounds is a daunting task that has withstood the efforts of many researchers over many years. If circuit lower bounds are indeed necessary to derandomize BPP, then no such derandomization results are likely to appear any time soon.

But, perhaps, BPP can be derandomized even in the absence of superpolynomial circuit lower bounds. While the existence of a quick PRG would imply a superpolynomial circuit lower bound for EXP (see Theorem 1), no such lower bound is known to be implied by the assumption $\mathbf{BPP} = \mathbf{P}$, or even by the stronger assumption that the acceptance probability of a given Boolean circuit can be approximated in deterministic polynomial time (see also [KRC00, For01] for further discussion).

However, Impagliazzo, Kabanets, and Wigderson [IKW02] show that the existence of a non-deterministic subexponential-time algorithm for approximating the circuit acceptance probability would imply a superpolynomial circuit lower bound for $\text{NEXP} = \text{NTIME}(2^{\text{poly}(n)})$. In fact, they prove an even stronger result saying that it is impossible to separate NEXP and MA without proving that $\text{NEXP} \not\subseteq \text{P/poly}$.

**Theorem 12 ([IKW02]).** If $\text{NEXP} \subseteq \text{P/poly}$, then $\text{NEXP} = \text{MA}$. 

11
Proof Sketch. Since $\text{EXP} \subset \text{P}/\text{poly}$ implies $\text{EXP} = \text{MA}$ [BFL90], it will be sufficient to prove that $\text{NEXP} \subset \text{P}/\text{poly}$ implies $\text{NEXP} = \text{EXP}$.

We use the “easy witness” generator $\text{Easy} : \{0,1\}^{\text{poly}(n)} \rightarrow \{0,1\}^{2^n}$, defined in Section 4.2, to search for $\text{NEXP}$-witnesses. If this generator succeeds for all $\text{NEXP}$ languages, then $\text{NEXP} = \text{EXP}$, and we are done. The rest of the proof argues that $\text{Easy}$ must succeed.

Suppose otherwise. Then there is a $\text{NEXP}$ Turing machine $M$ for which $\text{Easy}$ fails. Using $M$, we can nondeterministically guess $n$-input Boolean functions of circuit complexity greater than $n^c$, for any $c > 0$.

Indeed, let $x \in \{0,1\}^n$ be such that $x \in L(M)$ but $\text{Easy}$ failed to find any $\text{NEXP}$-witness for $x$. Then, using $x$ as an advice string, we can guess a $\text{NEXP}$-witness for $x$ which must be the truth table of a hard Boolean function since, otherwise, $\text{Easy}$ would have found this witness. If $\text{NEXP} \neq \text{EXP}$, there will be infinitely many such advice strings $x$, and so there will be infinitely many $n$ such that we can guess $n$-input Boolean functions of high circuit complexity. Also note that the advice strings of size $n$ enable us to guess $n$-input Boolean functions of hardness greater than $n^c$ for any $c > 0$.

Plugging these hard Boolean functions into the known hardness-randomness tradeoffs implies that $\text{MA}$ is in nondeterministic subexponential time, for infinitely many input lengths, and using sublinear advice. Our assumption that $\text{NEXP} \subset \text{P}/\text{poly}$ can then be used to show the existence of some universal constant $c_0$ such that every language in $\text{MA}$ can be computed by Boolean circuits of size $n^{c_0}$, infinitely often.

Recall that, under our assumption that $\text{NEXP} \subset \text{P}/\text{poly}$, we have $\text{EXP} = \text{MA}$. Thus, we conclude that every language in $\text{EXP}$ can be computed by circuits of size $n^{c_0}$, infinitely often. But this is impossible by a simple diagonalization argument.

It follows from [BFT98] that Theorem 12 does not relativize.

5.2 Circuit lower bounds from the derandomization of $\text{BPP}$

As noted earlier, no Boolean circuit lower bounds for $\text{EXP}$, or even for $\text{NEXP}$, are known to follow from the assumption that $\text{BPP} = \text{P}$. However, as recently shown by Impagliazzo and Kabanets [KI03], $\text{BPP} = \text{P}$ implies either Boolean circuit lower bounds for $\text{NEXP}$ or algebraic circuit lower bounds for the Permanent function. Recall that the permanent of an $n \times n$ integer matrix $A = (a_{i,j})$ is $\sum_{\sigma} \prod_{i=1}^{n} a_{i,\sigma(i)}$, where the summation is over all permutations $\sigma$ of $\{1,\ldots,n\}$.

The main result in [KI03] actually shows that derandomizing a specific $\text{BPP}$ problem, Polynomial Identity Testing, is essentially equivalent to proving circuit lower bounds for $\text{NEXP}$.

Theorem 13 ([KI03]). If one can test in polynomial time (or, even, nondeterministic subexponential time, infinitely often) whether a given arithmetic circuit over integers computes an identically zero polynomial, then either

1. $\text{NEXP} \not\subset \text{P}/\text{poly}$, or
2. Permanent is not computable by polynomial-size arithmetic circuits.

Proof Sketch. Our proof is by contradiction. The main observation is that the Permanent of an $n \times n$ matrix $A$ is downward self-reducible via the expansion by minors formula:

$$\text{perm}(A) = \sum_{j=1}^{n} a_{1,j} \ast \text{perm}(A|j), \quad (1)$$
where $A[j]$ denotes the submatrix of $A$ obtained by removing the first row and the $j$th column. Viewing $A$ as a matrix of integer variables, equality (1) is just a polynomial identity. By our assumption, we can test polynomial identities in polynomial time.

Now suppose that $C$ is an arithmetic circuit that purports to compute the permanent of $n \times n$ integer matrices. Then by testing whether $C$ satisfies $n$ identities of type (1) (one for each matrix size from 1 to $n$), we can test if $C$ is indeed a correct circuit. Assuming that Permanent is computable by polynomial-size arithmetic circuits implies that Permanent is in NP: we can guess a small arithmetic circuit, and then verify the correctness of our guess (by testing a small number of polynomial identities).

To conclude the proof, note that, by Theorem 12, if NEXP $\subseteq$ P/poly, then NEXP = MA. The class MA is contained in the second level of the polynomial-time hierarchy, and hence, by the results of Toda [Tod91] and Valiant [Val79], it can be simulated in P$^{\text{Permanent}}$. But, as we just observed, if Permanent is computable by polynomial-size arithmetic circuits, then Permanent is in NP, and so, we get NEXP = MA = NP, which is impossible by diagonalization.

The following is a partial converse of Theorem 13.

**Theorem 14 ([KI03]).** If Permanent cannot be computed by polynomial-size arithmetic circuits, then one can test in subexponential time, infinitely often, whether a given arithmetic formula computes an identically zero polynomial.

### 6 Other Results

Using hardness-randomness tradeoffs, Cai, Nerurkar, and Sivakumar [CNS99] prove a tight time-hierarchy theorem for the class BPQP = BPTIME($2^{\text{polylog}(n)}$), under the assumption that EXP contains a language of circuit complexity $2^{n^{O(1)}}$ or that PERMANENT $\not\in \cap_{\epsilon > 0} \text{BPTIME}(2^{n^\epsilon})$.

Klivans and van Melkebeek [KM99] prove a hardness-randomness tradeoff for space-bounded computation. In particular, they show that BPL = L if there is a language in LNSPACE that requires branching programs of size $2^{\Omega(n)}$; here, BPL is the class of languages accepted by logspace randomized Turing machines with bounded two-sided error. This answers a question from [CRT98].

Raz and Reingold [RR99] obtain improved derandomization results for certain restricted classes of space-bounded computation.

### 7 What Next?

An interesting open problem is to extend the uniform hardness-randomness tradeoff, Theorem 6, to other time bounds. For example, does the assumption EXP $\not\subseteq \cap_{\epsilon > 0} \text{BPTIME}(2^{n^\epsilon})$ imply that, in the “uniform setting”, BPP $\subseteq \text{DTIME}(2^{\text{polylog}(n)})$ infinitely often? Also, does Theorem 6 relativize?

Another problem is to decide if circuit lower bounds for EXP (rather than NEXP) are needed for the derandomization of BPP or promiseBPP. It is still an open question whether any Boolean circuit lower bounds for NEXP are implied by the assumption BPP = P.

The main open problem is, of course, the old one: prove an unconditional derandomization result for BPP or ZPP. In view of the results in Section 5, derandomizing the class BPP is quite hard. Potentially, it may be easier to derandomize ZPP, as there are no known circuit lower bounds implied by the assumption that ZPP = P.
Acknowledgments  I want to thank Lance Fortnow, Oded Goldreich, Russell Impagliazzo, Dieter van Melkebeek, Chris Umans, Salil Vadhan, and Avi Wigderson for a number of helpful comments and suggestions that significantly improved the quality of this presentation.

References


