The Computational Complexity Column

by

Lance FORTNOW

NEC Research Institute
4 Independence Way, Princeton, NJ 08540, USA
fortnow@research.nj.nec.com
http://www.neci.nj.nec.com/homepages/fortnow/beatcs

Perhaps no area in computational complexity has seen greater progress over the past several years than derandomization, i.e., the ability to remove randomness from probabilistic computation. The last derandomization survey in these pages dates only four years ago and yet we have seen quite a big of progress since then. Valentine Kabanets takes us through the recent results in derandomization.

Derandomization: A Brief Overview

Valentine Kabanets[1]
Department of Computer Science
University of California, San Diego
La Jolla, CA 91097-0114
kabanets@cs.ucsd.edu

Abstract

This survey focuses on the recent[2] 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 The class BPP has effectively superseded the class P as the class of problems that are considered efficiently solvable, while many researchers believe that BPP = P.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]Supported by a Postdoctoral Fellowship from the Natural Sciences and Engineering Research Council of Canada.

[2]I have chosen 1998 as a cut-off year for the present survey so as to complement the most recent survey on derandomization [CRT98] that appeared in this Bulletin.

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 $\mathsf{EXP} = \mathsf{DTIME}(2^{\mathsf{poly}(n)})$ 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 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 EXP contains a language of superpolynomial circuit complexity (i.e., $\mathsf{EXP} \not\subset \mathsf{P/poly}$), 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 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 $\mathsf{BPP} = \mathsf{P}$ 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 $\mathsf{BPP} = \mathsf{P}$, 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 E contains a language of circuit complexity $2^{\Omega(n)}$ almost everywhere, then $\mathsf{BPP} = \mathsf{P}$.

1.1.4 Hitting-set generators

Independently of [IW97], and using different methods, Andreev, Clementi, and Rolim [ACR97], based on [ACR98], showed that $\mathsf{BPP} = \mathsf{P}$ 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 RP algorithm, rather than BPP algorithm. Somewhat surprisingly, the existence of an efficient hitting-set generator implies that $\mathsf{BPP} = \mathsf{P}$ [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. The readers interested in a more detailed presentation are referred to the excellent survey on derandomization by 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 $1 - \frac{1}{\mathsf{poly}(n)}$ 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 that $\frac{1}{2}+\frac{1}{2^{\Omega(n)}}$ 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 $\frac{1}{2}+\frac{1}{2^{\Omega(n)}}$ 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 $\frac{1}{2}+\frac{1}{2^{\Omega(n)}}$ 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 **Enc**, a given message string m of length N into a codeword string $c=\mathbf{Enc}(m)$ of bigger length so that the original message m can be efficiently recovered, via the decoding function **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 $c=\mathbf{Enc}(m)$ 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 **Enc** and **Dec**, with the following properties:

1. $|\mathbf{Enc}(m)|\in\mathsf{poly}(|m|)$,

2. the message m can be recovered from a string r whenever r and $c=\mathbf{Enc}(m)$ agree in at least $\frac{1}{2}+\frac{1}{|c|^{\Omega(1)}}$ 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 $G : \{0,1\}^n \to \{0,1\}^m$, 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 m inputs,

$$|\mathbf{Pr}_x[C(x)=1] - \mathbf{Pr}_y[C(G(y))=1]| < \frac{1}{m},$$

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 $f : \{0,1\}^n \to \{0,1\}$ 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 $G : \{0,1\}^n \to \{0,1\}^m$, then there is a $2^{O(n)}$-time computable Boolean function $f : \{0,1\}^{n+1} \to \{0,1\}$ 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 $p = \mathbf{Pr}_x[f(x)=1] \geqslant \frac{1}{2}$, 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 $\mathbf{Pr}_y[f(G(y))=1] = 0 < p - 1/m$, for any $m > 2$. Thus, f must have hardness at least m. □

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, GVW00], 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 $f : \{0,1\}^n \to \{0,1\}$ of hardness $s = s(n)$, then $\mathsf{BPTIME}(t) \subseteq \mathsf{DTIME}(2^{O(s^{-1}(t^{O(1)}))})$, 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 [Uma01] obtains the following *optimal* hardness-randomness tradeoff for *PRGs*, which also implies Theorem 2.

Theorem 3 ([Uma01]). *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 $L \in \mathsf{MA}$ if there is a polynomial-time computable relation R_L such that, for any string x,

$$x \in L \Rightarrow \exists y \ : \ \mathbf{Pr}_z[R_L(x,y,z) = 1] > 3/4,$$
$$x \notin L \Rightarrow \forall y \ : \ \mathbf{Pr}_z[R_L(x,y,z) = 1] < 1/4,$$

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 $\mathbf{Pr}_z[R_L(x,y,z) = 1]$ deterministically in polynomial time, and hence imply that $\mathsf{MA} \subseteq \mathsf{NP}$. Thus, the known hardness-randomness tradeoffs show that the existence of a language in $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$ of high circuit complexity implies the derandomization of MA.

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

$$x \in L \Rightarrow \mathbf{Pr}_z[\exists y \ : \ R_L(x,y,z) = 1] > 3/4,$$
$$x \notin L \Rightarrow \mathbf{Pr}_z[\exists y \ : \ R_L(x,y,z) = 1] < 1/4,$$

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

To derandomize AM, we would need to estimate the acceptance probability of a *nondeterministic* 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 $f(x,z) = 1 \Leftrightarrow \exists y : R_L(x,y,z) = 1$ is in P^{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 $\mathsf{NE} = \mathsf{NTIME}(2^{O(n)})$.

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

Miltersen and Vinodchandran [MV99] improve upon Theorem 4 by replacing the assumption of high SAT-oracle circuit complexity with that of high *nondeterministic* 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, Uma01].

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* $2^{\Omega(n)}$ *almost everywhere, then the following task can be performed deterministically in polynomial time: given a propositional formula* ϕ*, generate a list of propositional formulas such that*

- *if* ϕ *is unsatisfiable, then so is every formula on the list, and*
- *if* ϕ *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 $T_0 : \{0,1\}^k \to \{0,1\}$ 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 $T_0 : \{0,1\}^k \to \{0,1\}$ 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; the description of this research deserves a separate 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, Uma01] 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 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 $\mathsf{EXP} \not\subseteq \mathsf{P}$ should yield a

PRG for any 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* $\mathsf{EXP} \not\subseteq \mathsf{BPP}$, *then, for every* $\epsilon > 0$, *there is a quick generator* $G : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ *that is pseudorandom with respect to any* P*-samplable family of* n*-size Boolean circuits infinitely often.*

The phrase "G is pseudorandom with respect to any P-samplable 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

$$|\mathbf{Pr}_x[C(x) = 1] - \mathbf{Pr}_y[C(G(y)) = 1]| \geqslant \frac{1}{n}.$$

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

$$\mathbf{Pr}[R(1^n) \in B_G(n)] < \frac{1}{n},$$

where the probability is over the internal coin tosses of R.

Proof Sketch of Theorem 6. If $\mathsf{EXP} \not\subset \mathsf{P/poly}$, then Theorem 6 follows by the standard (nonuniform) hardness-randomness tradeoff from [BFNW93]. On the other hand, if $\mathsf{EXP} \subset \mathsf{P/poly}$, then EXP collapses to Σ_2^p [KL82], and since $\Sigma_2^p \subseteq \mathsf{P}^{\#\mathsf{P}}$ [Tod91], we conclude that $\#\mathsf{P}$-complete languages are also complete for EXP. Thus, it suffices to consider a generator based on PERMANENT [Val79].

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

The fact that 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 PERMANENT_n, we first construct small circuits $C_1, \dots, C_{n-1}$ computing $\text{PERMANENT}_1, \dots, \text{PERMANENT}_{n-1}$, 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 PERMANENT_n. This shows that PERMANENT is in BPP, and hence, $\mathsf{EXP} = \mathsf{BPP}$. $\square$

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

Theorem 7 ([IW98]). *If* $\mathsf{EXP} \neq \mathsf{BPP}$, *then, for any* $\epsilon > 0$, *every* BPP *algorithm can be simulated deterministically in time* 2^{n^ϵ} *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 $\mathsf{EXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \Sigma_2^p$. It is not known, however, whether Theorem 6 itself relativizes.

4.2 Derandomizing RP

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

For a generator $H : \{0,1\}^k \to \{0,1\}^n$, let $B_H(n)$ be the set of all circuits C of size n such that $\mathbf{Pr}_x[C(x) = 1] \geqslant 1/2$ but $\mathbf{Pr}_y[C(H(y)) = 1] = 0$; 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*-samplable 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

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

Theorem 8 ([Kab00]). *If* $\mathsf{EXP} \not\subseteq \mathsf{ZPP}$*, then, for every* $\epsilon > 0$*, there is a quick generator* $H : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ *that is a hitting-set generator with respect to any* P*-samplable family of* n*-size Boolean circuits infinitely often.*

The proof of Theorem 8 uses the "easy witness" generator $\mathrm{Easy} : \{0,1\}^k \to \{0,1\}^n$ defined as follows. For any $y \in \{0,1\}^k$, $\mathrm{Easy}(y) = t$ 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 $\mathrm{Easy} : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ can be uniformly broken for some $\epsilon > 0$, then $\mathsf{BPP} = \mathsf{ZPP}$.

Indeed, suppose that the generator Easy is *not* a hitting-set generator with respect to some P-samplable 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^ϵ. Consequently, we can probabilistically guess, with *zero error*, a hard string and convert it into pseudorandomness via the known hardness-randomness tradeoffs. The conclusion $\mathsf{BPP} \subseteq \mathsf{ZPP}$ follows.

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

A corollary of Theorem 8 is the following *unconditional* result about the "easiness" of RP in a certain uniform setting.

Theorem 9 ([Kab00]). *At least one of the following holds.*

1. $\mathsf{RP} \subseteq \mathsf{ZPP}$.
2. *For any* $\epsilon > 0$*, every* RP *algorithm can be simulated in deterministic time* 2^{n^ϵ} *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* $f(1^n)$*.*

4.3 Derandomizing the Graph Nonisomorphism Problem

Lu [Lu00] considers the modified generator $\mathrm{Easy}_{SAT} : \{0,1\}^k \to \{0,1\}^n$ 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 Easy_{SAT} to search for NP-*witnesses*, i.e., checking if any output of Easy_{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. $\mathsf{AM} \subseteq \mathsf{NP}$.
2. *For any $\epsilon > 0$, every* NP *(and every* coNP*) algorithm can be simulated in deterministic time 2^{n^ϵ} 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 $f : \{1\}^n \to \{0,1\}^n$.

5 Hitting the Wall?

Hardness-randomness tradeoffs have been hailed as a step forward in the quest to prove that $\mathsf{BPP} = \mathsf{P}$: 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 the superpolynomial circuit lower bound for EXP (see Theorem 1), no such lower bound is known to be implied by the assumption $\mathsf{BPP} = \mathsf{P}$, 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 [IKW01] show that the existence of a nondeterministic subexponential-time algorithm for approximating the circuit acceptance probability would imply the superpolynomial circuit lower bound for $\mathsf{NEXP} = \mathsf{NTIME}(2^{\mathsf{poly}(n)})$. In fact, they prove an even stronger result saying that it is *impossible* to separate NEXP and MA *without* proving that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.

Theorem 11 ([IKW01]). *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then* $\mathsf{NEXP} = \mathsf{MA}$*.*

Proof Sketch. Since $\mathsf{EXP} \subset \mathsf{P/poly}$ implies $\mathsf{EXP} = \mathsf{MA}$ [BFL91], it will be sufficient to prove that $\mathsf{NEXP} \subset \mathsf{P/poly}$ implies $\mathsf{NEXP} = \mathsf{EXP}$.

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

Suppose otherwise. Then there is a NEXP Turing machine M for which 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 Easy failed to find any NEXP-witness for x. Then, using x as an advice string, we can guess a NEXP-witness for x which must be the truth table of a hard Boolean function since, otherwise, Easy would have found this witness. If $\mathsf{NEXP} \neq \mathsf{EXP}$, 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 MA is in nondeterministic subexponential time, for infinitely many input lengths, and using sublinear advice. Our assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$ can then be used to show the existence of some universal constant c_0 such that every language in MA can be computed by Boolean circuits of size n^{c_0}, infinitely often.

Recall that, under our assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$, we have $\mathsf{EXP} = \mathsf{MA}$. Thus, we conclude that every language in 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 11 does not relativize.

As noted earlier, no circuit lower bounds for EXP are known to follow from the assumption that $\mathsf{BPP} = \mathsf{P}$. Such an implication would be immediate if we could show that $\mathsf{BPEXP} \not\subset \mathsf{P/poly}$, where BPEXP is the exponential-time version of BPP. Indeed, assume that $\mathsf{BPEXP} \not\subset \mathsf{P/poly}$. If $\mathsf{BPP} = \mathsf{P}$, then $\mathsf{BPEXP} = \mathsf{EXP}$ by padding, and hence, by our assumption, $\mathsf{EXP} \not\subset \mathsf{P/poly}$.

On the other hand, the superpolynomial lower bound for $\mathsf{MA\text{-}EXP}$, the exponential-time version of MA, is known [BFT98]. Curiously, both the proof of the lower bound for $\mathsf{MA\text{-}EXP}$ and the proof of Theorem 11 heavily depend on the same result from [BFL91]: $\mathsf{EXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \mathsf{MA}$.

6 Other Results

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

Klivans and van Melkebeek [KM99] prove a hardness-randomness tradeoff for *space*-bounded computation. In particular, they show that $\mathsf{BPL} = \mathsf{L}$ if there is a language in $\mathsf{LINSPACE}$ 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 $\mathsf{EXP} \not\subseteq \cap_{\epsilon>0}\mathsf{BPTIME}(2^{n^\epsilon})$ imply that, in the "uniform setting", $\mathsf{BPP} \subseteq \mathsf{DTIME}(2^{\mathsf{polylog}(n)})$ infinitely often? Also, does Theorem 6 relativize?

Another problem is to decide if circuit lower bounds for EXP are needed for the derandomization of BPP or promiseBPP. If true, can the necessity of such lower bounds be proved without showing $\mathsf{BPEXP} \not\subset \mathsf{P/poly}$ along the way?

The main open problem is, of course, the old one: prove an unconditional derandomization result for BPP or ZPP.

Acknowledgments I want to thank Lance Fortnow, Russell Impagliazzo, Dieter van Melkebeek, Chris Umans, and Salil Vadhan for a number of helpful comments and suggestions that significantly improved the quality of this presentation.

References

[ACR97] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Worst-case hardness suffices for derandomization: A new method for hardness vs. randomness trade-offs. In *Proceedings of the Twenty-Fourth International Colloquium on Automata, Languages, and Programming*, pages 177–187, 1997.

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the Association for Computing Machinery*, 45(1):179–213, 1998. (preliminary version in ICALP'96).

[ACRT99] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999. (preliminary version in FOCS'97).

[AK97] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. In *Proceedings of the Seventeenth Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *Lecture Notes in Computer Science*, pages 235–249. Springer Verlag, 1997.

[AS97] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, 1997.

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48, Berlin, 1990. Springer Verlag.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In C. Meinel and S. Tison, editors, *Proceedings of the Sixteenth Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer Verlag, 1999.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

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

[BFT98] H. Buhrman, L. Fortnow, and L. Thierauf. Nonrelativizing separations. In *Proceedings of the Thirteenth Annual IEEE Conference on Computational Complexity*, pages 8–12, 1998.

[BH89] R. Boppana and R. Hirschfeld. Pseudo-random generators and complexity classes. In S. Micali, editor, *Advances in Computing Research*, volume 5, pages 1–26. JAI Press, 1989.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[CNS99] J.-Y. Cai, A. Nerurkar, and D. Sivakumar. Hardness and hierarchy theorems for probabilistic quasi-polynomial time. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 726–735, 1999.

[CRT98] A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan. Recent advances towards proving P=BPP. *Bulletin of the European Association for Theoretical Computer Science*, (64):96–103, February 1998.

[For01] L. Fortnow. Comparing notions of full derandomization. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 28–34, 2001.

[GKL88] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudo-random generators. In *Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 12–24, 1988.

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

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the Association for Computing Machinery*, 38:691–729, 1991.

[GNW95] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR-Lemma. *Electronic Colloquium on Computational Complexity*, TR95-050, 1995.

[GS89] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Advances in Computing Research*, volume 5, pages 73–90. JAI Press, 1989.

[GVW00] O. Goldreich, S. Vadhan, and A. Wigderson. Simplified derandomization of BPP using a hitting set generator. *Electronic Colloquium on Computational Complexity*, TR00-004, 2000.

[GW99] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In D. Hochbaum, K. Jansen, J.D.P. Rolim, and A. Sinclair, editors, *Randomization, Approximation, and Combinatorial Optimization*, volume 1671 of *Lecture Notes in Computer Science*, pages 131–137. Springer Verlag, 1999. (RANDOM-APPROX'99).

[GZ97] O. Goldreich and D. Zuckerman. Another proof that BPP$\subseteq$PH (and more). *Electronic Colloquium on Computational Complexity*, TR97-045, 1997.

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

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 1–11, 2001.

[Imp95] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the Thirty-Sixth Annual IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 181–190, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudorandom generators with optimal seed length. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 1–10, 2000.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

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

[Kab00] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Proceedings of the Fifteenth Annual IEEE Conference on Computational Complexity*, pages 150–157, 2000.

[KL82] R.M. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(3-4):191–209, 1982. (preliminary version in STOC'80).

[KM99] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 659–667, 1999.

[KRC00] V. Kabanets, C. Rackoff, and S. Cook. Efficiently approximable real-valued functions. *Electronic Colloquium on Computational Complexity*, TR00-034, 2000.

[Lev87] L.A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merrit, editors, *Distributed Computing and Cryptography*, pages 191–202. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 2, AMS, 1991.

[Lu00] C.-J. Lu. Derandomizing Arthur-Merlin games under uniform assumptions. In *Proceedings of the Eleventh Annual International Symposium on Algorithms and Computation (ISAAC'00)*, 2000.

[Mil01] P.B. Miltersen. Derandomizing complexity classes. In S. Rajasekaran P. Pardalos, J. Reif, and J. Rolim, editors, *Handbook of Randomized Computing*, volume II. Kluwer Academic Publishers, 2001. (a draft is available at `www.brics.dk/~bromille`).

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, 1995.

[MV99] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 71–80, 1999.

[Nis91] N. Nisan. Pseudo random bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

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

[Plo60] M. Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6:445–450, 1960.

[RR99] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 168–178, 1999.

[Sha81] A. Shamir. On the generation of cryptographically strong pseudo-random sequences. In *Proceedings of the Eighth International Colloquium on Automata, Languages, and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 544–550. Springer Verlag, 1981.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. (preliminary version in STOC'99).

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the Forty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[Sud97] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Sud00] M. Sudan. List decoding: Algorithms and applications. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *Proceedings of the International Conference IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*, pages 25–41. Springer Verlag, August 2000.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Tre99] L. Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 141–148, 1999.

[TSZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the Forty-Second Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[Uma01] C. Umans. Pseudo-random generators for all hardnesses. manuscript (submitted), November 2001.

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

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

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.