Pseudorandom generators without the XOR Lemma*

Madhu Sudan[†] Luca Trevisan[‡] Salil Vadhan[§]

May 18, 2000

Abstract

Impagliazzo and Wigderson [IW97] have recently shown that if there exists a decision problem solvable in time $2^{O(n)}$ and having circuit complexity $2^{\Omega(n)}$ (for all but finitely many n) then $\mathsf{P} = \mathsf{BPP}$. This result is a culmination of a series of works showing connections between the existence of hard predicates and the existence of good pseudorandom generators.

The construction of Impagliazzo and Wigderson goes through three phases of "hardness amplification" (a multivariate polynomial encoding, a first derandomized XOR Lemma, and a second derandomized XOR Lemma) that are composed with the Nisan–Wigderson [NW94] generator. In this paper we present two different approaches to proving the main result of Impagliazzo and Wigderson. In developing each approach, we introduce new techniques and prove new results that could be useful in future improvements and/or applications of hardness-randomness trade-offs.

Our first result is that when (a modified version of) the Nisan-Wigderson generator construction is applied with a "mildly" hard predicate, the result is a generator that produces a distribution indistinguishable from having large min-entropy. An extractor can then be used to produce a distribution computationally indistinguishable from uniform. This is the first construction of a pseudorandom generator that works with a mildly hard predicate without doing hardness amplification.

We then show that in the Impagliazzo–Wigderson construction only the first hardness-amplification phase (encoding with multivariate polynomial) is necessary, since it already gives the required average-case hardness. We prove this result by (i) establishing a connection between the hardness-amplification problem and a list-decoding problem for error-correcting codes; and (ii) presenting a list-decoding algorithm for error-correcting codes based on multivariate polynomials that improves and simplifies a previous one by Arora and Sudan [AS97].

Keywords: Pseudorandom generators, extractors, polynomial reconstruction, list decoding

---

*Preliminary versions and abstracts of this paper appeared in ECCC [STV98], STOC '99 [STV99b], and Complexity '99 [STV99a].

[†]Laboratory for Computer Science, 545 Technology Square, MIT, Cambridge, MA 02141. E-mail: madhu@theory.lcs.mit.edu. Research supported in part by a Sloan Foundation Fellowship, an MIT-NEC Research Initiation Grant and NSF Career Award C-CR-9875511.

[‡]Department of Computer Science, Columbia University, 500W 120th St., New York, NY 10027. Email: luca@cs.columbia.edu. Work done at MIT.

[§]Laboratory for Computer Science, 545 Technology Square, MIT, Cambridge, MA 02141. E-mail: salil@theory.lcs.mit.edu. URL: http://theory.lcs.mit.edu/~salil. Work done while supported by a DOD/NDSEG graduate fellowship and partially by DARPA grant DABT63-96-C-0018.

1 Introduction

This paper continues the exploration of hardness versus randomness trade-offs, that is, results showing that randomized algorithms can be efficiently simulated deterministically if certain complexity-theoretic assumptions are true. We present two new approaches to proving the recent result of Impagliazzo and Wigderson [IW97] that, if there is a decision problem computable in time $2^{O(n)}$ and having circuit complexity $2^{\Omega(n)}$ then $\mathsf{P} = \mathsf{BPP}$. Impagliazzo and Wigderson prove their result by presenting a "randomness-efficient amplification of hardness" based on a derandomized version of Yao's XOR Lemma. The hardness-amplification procedure is then composed with the Nisan–Wigderson (NW) generator [NW94] to yield the result. The hardness amplification goes through three steps: an encoding using multivariate polynomials (from [BFNW93]), a first derandomized XOR Lemma (from [Imp95]) and a second derandomized XOR Lemma (which is the technical contribution of [IW97]).

In our first result, we show how to construct a "pseudoentropy generator" starting from a predicate with "mild" hardness. Roughly speaking, a *pseudoentropy generator* takes a short random seed as input and outputs a distribution that is indistinguishable from having high min-entropy. Combining our pseudoentropy generator with an *extractor*, we obtain a pseudorandom generator. Interestingly, our pseudoentropy generator is (a modification of) the NW generator itself. Along the way we make the new observation that, when built out of a mildly hard predicate, the NW generator outputs a distribution that is indistinguishable from having high Shannon entropy. The notion of a pseudoentropy generator, and the idea that a pseudoentropy generator can be converted into a pseudorandom generator using an extractor, are due to Håstad et al. [HILL99].[1] Our construction is the first construction of a pseudorandom generator that directly uses a mildly hard predicate without hardness amplification.

We then revisit the hardness amplification problem, as considered in [BFNW93, Imp95, IW97], and we show that the first step alone (encoding with multivariate polynomials) is sufficient to amplify hardness to the desired level, so that the derandomized XOR Lemmas are not necessary in this context. Our proof is based on a list-decoding algorithm for multivariate polynomial codes and exploits a connection between the list-decoding and the hardness-amplification problems. The list-decoding algorithm described in this paper is quantitatively better than a previous one by Arora and Sudan [AS97], and has a simpler analysis.

An overview of previous results. The works of Blum and Micali [BM84] and Yao [Yao82] formalize the notion of a pseudorandom generator and show how to construct pseudorandom generators based on the existence of one-way permutations. A pseudorandom generator meeting their definitions (which we call a *BMY-type PRG*) is a polynomial-time algorithm that on input a randomly selected string of length n^ϵ produces an output of length n that is computationally indistinguishable from uniform by any adversary of $\mathrm{poly}(n)$ size, where ϵ is an arbitrarily small constant. Pseudorandom generators of this form and "pseudorandom functions" [GGM86] constructed from them have many applications both inside and outside cryptography (see, e.g., [GGM86, Val84, RR97]). One of the first applications, observed by Yao [Yao82], was to derandomization — any given polynomial-time randomized algorithm can be simulated deterministically using a BMY-type PRG in time $2^{n^\epsilon} \cdot \mathrm{poly}(n)$ by trying all the seeds and taking the majority answer.

In a seminal work, Nisan and Wigderson [NW94] explore the use of a weaker type of pseudorandom generator (PRG) in order to derandomize randomized algorithms. They observe that, for the purpose of derandomization, one can consider generators computable in time $\mathrm{poly}(2^t)$ (instead of $\mathrm{poly}(t)$) where t is the length of the seed, since the derandomization process cycles through all the seeds, and this induces an overhead factor 2^t anyway. They also observe that one can restrict to generators that are good against adversaries whose running time is bounded by a fixed polynomial, instead of every polynomial. They then show how to construct a pseudorandom generator meeting this relaxed definition under weaker assumptions than those used to build BMY-type pseudorandom generators. Furthermore, they show that, under a sufficiently strong assumption, one can build a PRG that uses seeds of logarithmic length (which would be impossible for a BMY-type PRG). Such a generator can be used to simulate randomized algorithms in polynomial time, and its existence implies $\mathsf{P} = \mathsf{BPP}$. The condition under which Nisan and Wigderson prove the existence of a PRG with seeds of logarithmic length is the existence of a decision problem (i.e., a predicate $P\colon \{0,1\}^n \to \{0,1\}$) solvable in time $2^{O(n)}$ such that for some positive constant ϵ no circuit of size

[1] To be accurate, the term *extractor* comes from [NZ96] and postdates the work of Håstad et al. [HILL99].

$2^{\epsilon n}$ can solve the problem on more than a fraction $1/2 + 2^{-\epsilon n}$ of the inputs (for all but finitely many n). This is a very strong hardness requirement, and it is of interest to obtain similar conclusions under weaker assumptions.

An example of a weaker assumption is the existence of a *mildly hard* predicate. We say that a predicate is mildly hard if for some fixed $\epsilon > 0$ no circuit of size $2^{\epsilon n}$ can decide the predicate on more than a fraction $1 - 1/\text{poly}(n)$ of the inputs. Nisan and Wigderson prove that mild hardness suffices to derive a pseudorandom generator with seeds of $\text{polylog}\, n$ length, which in turn implies a quasi-polynomial time deterministic simulation of BPP. This result is proved by using Yao's XOR Lemma [Yao82] (see, e.g., [GNW95] for a proof) to convert a mildly hard predicate over n inputs into one which has input size $\text{poly}(n)$ and is hard to compute on a fraction $1/2 + 2^{-\Omega(n)}$ of the inputs. A series of subsequent papers attacks the problem of obtaining stronger pseudorandom generators starting from weaker and weaker assumptions. Babai et al. [BFNW93] show that a predicate of *worst-case* circuit complexity $2^{\Omega(n)}$ can be converted into a mildly hard one.[2] Impagliazzo [Imp95] proves a derandomized XOR Lemma which implies that a mildly hard predicate can be converted into one that cannot be predicted on more than some *constant* fraction of the inputs by circuits of size $2^{\epsilon n}$. Impagliazzo and Wigderson [IW97] prove that a predicate with the latter hardness condition can be transformed into one that meets the hardness requirement of [NW94]. The result of [IW97] relies on a different derandomized version of the XOR Lemma than [Imp95]. Thus, the general structure of the original construction of Nisan and Wigderson [NW94] has been preserved in most subsequent works, progress being achieved by improving the single components. In particular, the use of an XOR Lemma in [NW94] continues, albeit in increasingly sophisticated forms, in [Imp95, IW97]. Likewise, the NW generator and its original analysis have always been used in conditional derandomization results since.[3] Future progress in the area will probably require a departure from this observance of the NW methodology, or at least a certain amount of revisitation of its main parts.

In this paper, we give two new ways to build pseudorandom generators with seeds of logarithmic length. Both approaches bypass the need for the XOR Lemma, and instead use tools (such as list decoding, extractors, and pseudoentropy generators) that did not appear in the sequence of works from [NW94] to [IW97]. For a diagram illustrating the steps leading up to the results of [IW97] and how our techniques depart from that framework, see Figure 1. Both of our approaches are described in more detail below.

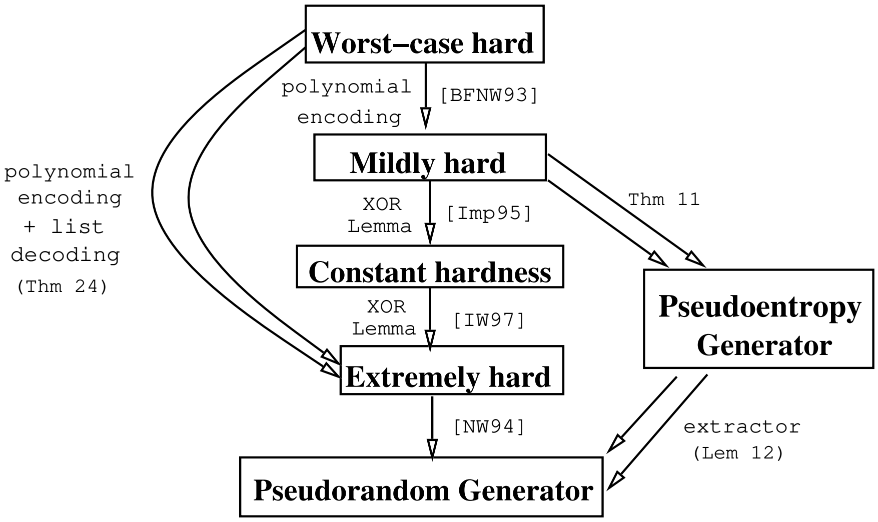

Figure 1: A comparison of our approach with previous ones. Double arrows indicate our results.

[2] In fact the result of [BFNW93] was somewhat weaker, but it is easily extendible to yield this result.
[3] The techniques of Andreev et al. [ACR97] are a rare exception, but they yield weaker results than those of [IW97].

A pseudoentropy generator. Nisan and Wigderson show that when their generator is constructed using a very hard-on-average predicate, then the output of the generator is indistinguishable from the uniform distribution. It is a natural question to ask what happens if there are stronger or weaker conditions on the predicate. In this paper we consider the question of what happens if the predicate is only mildly hard. Specifically we are interested in whether exponential average-case hardness is really necessary for direct pseudorandom generation. In this paper we first show that, when a mildly hard predicate is used in the NW generator, then there exists a distribution having high Shannon entropy that is indistinguishable from the output of the generator. Our main result is then that, for a mildly hard predicate, a modified version of the NW generator has an output indistinguishable from a distribution with high min-entropy. Such a generator is essentially a "pseudoentropy generator" in the sense of Håstad et al. [HILL99]. The intuition behind our proof starts with a result of Impagliazzo [Imp95] which says that if no small circuit can compute a predicate correctly on more than a fraction $1-\delta$ of the inputs, then there is some subset of the inputs of density δ on which the predicate is very hard on average. Due to the high hardness, the evaluation of the predicate in a random point of this set will be indistinguishable from a random bit. The NW generator constructed with a predicate P works by transforming an input seed s into a sequence of points $x_1,\ldots,x_m$ from the domain of P; the output of the generator is then $P(x_1)P(x_2)\cdots P(x_m)$. For a random seed, each of the points x_i is uniformly distributed, and so we expect to typically generate δm points from the hard set, so that the output of the generator looks like having δm bits of randomness, that is, it is indistinguishable from some other distribution having (Shannon) entropy δm. The generation of the points $x_1\cdots x_m$ can be modified so that the number of points landing in the hard set is sharply concentrated around its expected value δm. The output of the modified generator is then indistinguishable from having high min-entropy. When our generator is composed with a sufficiently good "extractor" (such as the one in [Tre99]) then the result is a pseudorandom generator. (An *extractor* is an algorithm that takes as input a string sampled from a distribution with high min-entropy, and produces as output a string that is statistically close to uniform. See Section 3.1 for a formal definition.)

This is the first construction of a pseudorandom generator based on mild average-case hardness that does not rely on hardness amplification. It is also the first application of the notion of a pseudoentropy generator to the construction of PRG in the Nisan–Wigderson sense.

Remark 1 While in this paper we analyze for the first time the Nisan-Wigderson generator under a *weaker* assumption than the one originally considered in [NW94], there has also been some work exploring the effect of *stronger* assumptions on the predicate. Impagliazzo and Wigderson [IW98] show that if the predicate has certain additional properties (such as "downward self-reducibility") then one needs only a *uniform* hardness assumption on the predicate (rather than a circuit-complexity assumption). Their conclusion is also weaker, obtaining only an average-case deterministic simulation of BPP for infinitely many input lengths. Arvind and Köbler [AK97] and Klivans and van Melkebeek [KvM99] show that if the predicate is hard on average for *nondeterministic* circuits, then the output of the generator is indistinguishable from uniform for nondeterministic adversaries. Therefore it is possible to derandomize classes involving randomness and nondeterminism, such as AM. Trevisan [Tre99] shows that if the predicate is chosen randomly from a distribution having certain properties, then the output is *statistically* close to uniform. This yields the construction of extractors that we use in our generator.

The connection with list decoding of error-correcting codes. Our second result deals with the "list-decoding problem" for error-correcting codes and its connection to amplification of hardness.

We start by describing a new "list-decoding" problem for error-correcting codes. This problem differs from the standard decoding task in that (1) the decoding algorithm is allowed to output a list of nearby codewords (rather than a unique nearest codeword) and (2) the decoding algorithm is allowed oracle access to the received word, and expected to decode in time much smaller than the length of the codeword. It is also allowed to output implicit representations of the list of codewords, by giving programs to compute the ith coordinate of each codeword. This implicit version of the list-decoding problem is closely related to and inspired by work in program checking and probabilistic checking of proofs.

We show a simple connection between amplification of hardness and the existence of (uniformly-constructible) families of codes with very efficient list-decoders in our sense (Theorem 24). We then show that a recent result of Arora and Sudan [AS97] on polynomial reconstruction leads to a family of error-correcting codes with very efficient list-decoders (Lemmas 25 and 28). In particular, this is sufficient to imply the hardness amplification results of [IW97]. Finally, we simplify the reconstruction procedure of Arora and Sudan and give an analysis (Theorem 29) that works for a wider range of parameters and has a much simpler proof. (In contrast, the analysis of Arora and Sudan relies on their difficult analysis of their "low-degree test" for the "highly noisy" case.)

The polynomial reconstruction problem has been studied for its applications to program checking, average-case hardness results for the permanent, and random self-reducibility of complete problems in high complexity classes [BF90, Lip89, GLR+91, FF93, GS92, FL96, CPS99]. The applicability of polynomial reconstruction to hardness versus randomness results was demonstrated by Babai et al. [BFNW93]. They show that the existence of a polynomial reconstruction procedure implies that one can convert a worst-case hard predicate into one which is mildly average-case hard by encoding it as a polynomial. In effect, our analysis shows that already at this stage the polynomial function is very hard, hard enough to use with the [NW94] pseudo-random generator. This connection between polynomial reconstruction and hardness amplification has also been observed independently by Avi Wigderson [Wig98] and S. Ravi Kumar and D. Sivakumar [KS98].

2 Preliminaries

Throughout the paper, all logarithms are with respect to base 2. We write U_n for the uniform distribution on $\{0,1\}^n$. Let X and Y be random variables on a discrete universe $\mathcal{U}$, and let $S : \mathcal{U} \to \{0,1\}$ be any function/algorithm. We say that S *distinguishes* X and Y with *advantage* ε if $|\Pr[S(X)=1] - \Pr[S(Y)=1]| \geq \varepsilon$. The *statistical difference* between X and Y is the maximum advantage with which any function distinguishes them, i.e. $\max_{S \subset U} |\Pr[X \in S] - \Pr[Y \in S]|$.

Our main objects of study are pseudorandom generators:

Definition 2 *A function $G\colon \{0,1\}^d \to \{0,1\}^n$ is an (s,ε)* pseudorandom generator *if no circuit of size s can distinguish G from U_n with advantage greater than ε.*

Our results rely on the Nisan–Wigderson construction of pseudorandom generators, described below.

2.1 The Nisan–Wigderson generator

The combinatorial construction underlying the NW generator is a collection of sets with small intersections, called a *design*.

Lemma 3 (design [NW94, Tre99]) *For every $\ell, m \in \mathbb{N}$, there exists a family of sets $S_1, \ldots, S_m \subset \{1, \ldots, d\}$ such that*

1. $d = O\left(\frac{\ell^2}{\log m}\right)$,
2. *For all i, $|S_i| = \ell$, and*
3. *For all $i \neq j$ $|S_i \cap S_j| \leq \log m$,*

Moreover, such a family can be found deterministically in time $\mathrm{poly}(m, 2^d)$

For concreteness, one can think of $m = 2^{\theta \ell}$ for some small constant $\theta > 0$, so that $d = O(\ell) = O(\log m)$. Given such a family of sets, the NW generator takes a uniformly distributed string of length d and produces m strings of length ℓ. That is, given parameters ℓ and m, we take the family of sets given by Lemma 3 and define $\mathrm{NW}_{\ell,m}\colon \{0,1\}^d \to (\{0,1\}^\ell)^m$ by

$$\mathrm{NW}_{\ell,m}(x) = (x_{S_1}, x_{S_2}, \ldots, x_{S_m}),$$

where x_{S_i} denotes the projection of x onto the coordinates specified by S_i.

The key property of this generator used in [NW94, IW97] is that the strings x_{S_i} behave as if they are independent when they are used as inputs to a hard function. Let $P\colon \{0,1\}^\ell \to \{0,1\}$ be any predicate. Then the NW pseudorandom generator using P is a function $\text{NW-PRG}^P_{\ell,m}\colon \{0,1\}^d \to \{0,1\}^m$ given by

$$\text{NW-PRG}^P_{\ell,m}(x) = P(x_1)P(x_2)\cdots P(x_m), \qquad \text{where } (x_1,\ldots,x_m) = \text{NW}_{\ell,m}(x)$$

The main theorem of [NW94] is that if P is taken to be a sufficiently hard (on average) predicate, $\text{NW-PRG}^P_{\ell,m}$ is a good pseudorandom generator.

Theorem 4 ([NW94]) *Suppose $P\colon \{0,1\}^\ell \to \{0,1\}$ is a predicate such that no circuit of size s can compute P correctly on more than a fraction $\frac{1}{2}+\frac{\varepsilon}{m}$ of the inputs. Then,* $\text{NW-PRG}^P_{\ell,m}$ *is an $(s - O(m^2), \varepsilon)$ pseudorandom generator.*

The pseudorandom generators produced by this theorem can be spectacular, as the seed length $d = O(\ell^2/\log m)$ can be much smaller than (even logarithmic in) the number of output bits if P is sufficiently hard. The main drawback is that the hypothesis is also extremely strong (in that P must be very hard on average), and much work has been done to construct predicates that are strong enough for Theorem 4 based on weaker assumptions [BFNW93, Imp95, IW97, IW98]. In the next section, we analyze the quality of this generator when only a mildly hard predicate is used.

3 Pseudorandom generators via pseudoentropy

In this section, we show how to build a pseudorandom generator out of a mildly hard predicate in a different (and arguably more direct) way than [IW97]. Specifically, we show how to directly build a "pseudoentropy generator" from a mildly hard predicate and argue that applying an extractor to its output gives a pseudorandom generator.

3.1 Entropy, pseudoentropy, and extractors

The various types of entropy are measures for the amount of randomness in a probability distribution. If X is a random variable on a discrete universe $\mathcal{U}$, the *(Shannon) entropy* of X is defined to be

$$\mathrm{H}(X) = \mathop{\mathrm{E}}_{\alpha \leftarrow X}\left[\log \frac{1}{\Pr[X=\alpha]}\right].$$

In Section 3.2, we show that, when a mildly hard predicate is used in the Nisan–Wigderson pseudorandom generator, the output of the generator is indistinguishable from having high Shannon entropy. However, later we will need a stricter measure of entropy. The *min-entropy* of X is

$$\mathrm{H}_\infty(X) = \min_{\alpha \in \mathcal{U}}\left(\log \frac{1}{\Pr[X=\alpha]}\right).$$

In Section 3.3, we show how to modify the Nisan–Wigderson generator to obtain indistinguishability from high min-entropy. The following definition (following [HILL99]) formalizes the type of generator we obtain.

Definition 5 *A generator $G\colon \{0,1\}^d \to \{0,1\}^m$ is a (k,s,ε)* pseudoentropy generator *if there is a distribution D on $\{0,1\}^m$ of min-entropy at least k such that no circuit of size s can distinguish the output of G from D with advantage greater than ε.*

The above definition differs from that of [HILL99] in several ways. Most importantly, we require the output to be indistinguishable from having high min-entropy, whereas they only require that it be indistinguishable from having high Shannon entropy. They later convert the Shannon entropy to min-entropy by taking many samples on independent seeds, but we cannot afford the extra randomness needed to do this. Other differences are that we ask for indistinguishability against circuits rather than uniform adversaries, that we do not require that G be computable in polynomial time, and that we do not explicitly ask that k be larger than d (though the notion is uninteresting otherwise).

In Section 3.4, we will show that a pseudoentropy generator can be transformed into a pseudorandom generator using the following type of tool:

Definition 6 ([NZ96, NT99]) *A function* $\text{EXT}\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ *is a* (k, ε)-extractor *if for every distribution* D *on* $\{0,1\}^m$ *of min-entropy at least* k, $\text{EXT}(D, U_d)$ *has statistical difference at most* ε *from* U_n.

We will make use of the following recent construction of extractors:

Theorem 7 ([Tre99]) *For every* m, k, *and* ε *such that* $k \le m$, *there is a* (k, ε)*-extractor* $\text{EXT}\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^{\sqrt{k}}$ *such that*

$$d = O\left(\frac{\log^2(m/\varepsilon)}{\log k}\right)$$

and $\text{EXT}\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^{\sqrt{k}}$ *is computable in time* $\text{poly}(m, 2^d)$ *(and can be computed by a circuit of size* $\text{poly}(m, d)$*).*

3.2 Using a mildly hard predicate

Intuitively, the reason the NW pseudorandom generator works is that whenever x_i is a "hard instance" of P, $P(x_i)$ is indistinguishable from a random bit. If P is very hard as in the hypothesis of Theorem 4, then almost all inputs are hard instances. Thus, with high probability all the x_i's will be hard instances and the limited dependence of the x_i's guarantees that the $P(x_i)$'s will look simultaneously random.

Now suppose that P is instead only mildly hard, in the sense that no small circuit can compute it correctly on more than a $1-\delta$ fraction of inputs, for some small but noticeable δ. Intuitively, this means that some δ fraction of the inputs are extremely hard for P. Thus, we'd expect that a δ fraction of the output bits of $\text{NW-PRG}^P_{\ell,m}$ are indistinguishable from random, so that we should get some crude pseudorandomness out of the generator. In fact, this intuition about hard instances can be made precise, as shown by the following result of Impagliazzo [Imp95].

Theorem 8 (hardcore sets [Imp95]) *Suppose no circuit of size* s *can compute* $P\colon \{0,1\}^\ell \to \{0,1\}$ *on more than a* $1-\delta$ *fraction of the inputs in* $\{0,1\}^\ell$*. Then, for every* $\varepsilon > 0$*, there exists an* ε-hardcore set $H \subset \{0,1\}^\ell$ *such that* $|H| = \delta \cdot 2^\ell$ *and no circuit of size* $s' = \Omega(\varepsilon^2\delta^2 s)$ *can compute* P *correctly on more than a* $\frac{1}{2} + \varepsilon$ *fraction of the inputs in* H*.*

Using this theorem, we can prove something about the output of $\text{NW-PRG}^P_{\ell,m}$ when a mildly hard predicate P is used. Notice that if x is chosen uniformly at random, then each component $x_i = x_{S_i}$ of the output of $\text{NW}_{\ell,m}(x)$ is uniformly distributed in $\{0,1\}^\ell$. Hence, the *expected* number of x_i's that land in H is δm. Thus, the earlier intuition suggests that the output of $\text{NW-PRG}^P_{\ell,m}$ should have δm bits of pseudorandomness, and this is in fact true.

Theorem 9 *Suppose no circuit of size* s *can compute* $P\colon \{0,1\}^\ell \to \{0,1\}$ *on more than a* $1-\delta$ *fraction of the inputs in* $\{0,1\}^\ell$*. Then, for every* $\varepsilon > 0$*, there is a distribution* D *on* $\{0,1\}^m$ *of (Shannon) entropy at least* δm *such that no circuit of size* $s' = \Omega(\varepsilon^2/m^2)\cdot s - O(m^2)$ *can distinguish the output of* $\text{NW-PRG}^P_{\ell,m}\colon \{0,1\}^d \to \{0,1\}^m$ *from* D *with advantage greater than* ε*.*

Proof: Let H be a $(\varepsilon/\delta m)$-hardcore set for P, as given by Theorem 8. We will show that the following distribution satisfies the requirements of the theorem.

Distribution D: Choose x uniformly from $\{0,1\}^d$. Let $(x_1,\ldots,x_m) = \mathrm{NW}(x)$. If $x_i \in H$, select $b_i \in \{0,1\}$ uniformly at random; and if $x_i \notin H$, let $b_i = P(x_i)$. Output $b_1 \cdots b_m$.

First, we argue that the entropy of D is at least δm. Define $\#_H(x_1,\ldots,x_m)$ to be the number of x_i's that are in H. Then for any $x \in \{0,1\}^d$, the entropy of $D|_x$ (i.e., D conditioned on x) is $\#_H(\mathrm{NW}(x)))$. By the definition of the Nisan-Wigderson generator, each x_i is (individually) uniformly distributed and therefore lands in H with probability δ. By linearity of expectations, the expectation of $\#_H(\mathrm{NW}(x))$ (over uniformly selected x) is δm. Thus, since conditioning reduces entropy (cf., [CT91, Th. 2.6.5]),

$$
\begin{aligned}
\mathrm{H}(D) &\geq \mathop{\mathrm{E}}_x[\mathrm{H}(D|_x)] \\
&= \mathop{\mathrm{E}}_x[\#_H(\mathrm{NW}(x))] \\
&= \delta m
\end{aligned}
$$

Now we show that D and $\text{NW-PRG}^P_{\ell,m}$ are computationally indistinguishable. Suppose that some circuit C distinguishes the output of $\text{NW-PRG}^P_{\ell,m}$ from D with advantage greater than ε. We will show that C must be of size at least $\Omega(\varepsilon^2/m^2)\cdot s - O(m^2)$. By complementing C if necessary, we have

$$
\Pr\left[C(\text{NW-PRG}^P_{\ell,m}(U_d)) = 1\right] - \Pr\left[C(D) = 1\right] > \varepsilon.
$$

For $x \in \{0,1\}^\ell$ and $r \in \{0,1\}$, define

$$
Q(x,r) = \begin{cases} r & \text{if } x \in H \\ P(x) & \text{otherwise.} \end{cases}
$$

Now consider "hybrids" $D_0,\ldots,D_m$ of D and $\text{NW-PRG}^P_{\ell,m}(U_d)$ defined as follows:

Distribution D_i: Choose x uniformly from $\{0,1\}^d$ and choose $r_1,\ldots,r_m$ uniformly from $\{0,1\}$. For $j = 1,\ldots,m$, let $p_j = P(x_{S_j})$ and $q_j = Q(x_{S_j}, r_j)$ (where S_j and x_{S_j} are as in the definition of $\mathrm{NW}_{\ell,m}$). Output $q_1 \cdots q_i p_{i+1} \cdots p_m$.

Thus, $D_0 = \text{NW-PRG}^P(U_d)$, and $D_m = D$. By the "hybrid argument" of [GM84] (cf. [Gol95, Sec. 3.2.3]), there is an i such that

$$
\begin{aligned}
\varepsilon/m &< \Pr[C(D_{i-1}) = 1] - \Pr[C(D_i) = 1] \\
&= \delta \cdot \Pr[C(D_{i-1}) = 1 \mid x_{S_i} \in H] + (1-\delta)\Pr[C(D_{i-1}) = 1 \mid x_{S_i} \notin H] \\
&\quad - (\delta \cdot \Pr[C(D_i) = 1 \mid x_{S_i} \in H] + (1-\delta)\Pr[C(D_i) = 1 \mid x_{S_i} \notin H]) \\
&= \delta \cdot (\Pr[C(D_{i-1}) = 1 \mid x_{S_i} \in H] - \Pr[C(D_i) = 1 \mid x_{S_i} \in H]),
\end{aligned}
$$

where the last equality is because D_{i-1} and D_i are identical conditioned on $x_{S_i} \notin H$. Expanding and using the fact that $q_i = Q(x_{S_i}, r_i) = r_i$ when $x_{S_i} \in H$, we have

$$
\begin{aligned}
&\Pr_{x,r_i,\ldots,r_m}\left[C\left(P(x_{S_1})\cdots P(x_{S_{i-1}})r_i Q(x_{S_{i+1}}, r_{i+1})\cdots Q(x_{S_m}, r_m)\right) = 1 \mid x_{S_i} \in H\right] \\
&- \Pr_{x,r_{i+1},\ldots,r_m}\left[C\left(P(x_{S_1})\cdots P(x_{S_{i-1}})P(x_{S_i})Q(x_{S_{i+1}}, r_{i+1})\cdots Q(x_{S_m}, r_m)\right) = 1 \mid x_{S_i} \in H\right] > \frac{\varepsilon}{\delta m},
\end{aligned}
$$

where x is chosen uniformly in $\{0,1\}^d$ and $r_i,\ldots,r_m$ are selected uniformly in $\{0,1\}$. Renaming r_i as b and using the standard transformation from distinguishers to predictors [Yao82] (cf. [Gol99, Sec. 3.3.3]), we see that

$$
\Pr_{x,b,r_{i+1},\cdots,r_m}\left[C\left(P(x_{S_1})\cdots P(x_{S_{i-1}})bQ(x_{S_{i+1}}, r_{i+1})\cdots Q(x_{S_m}, r_m)\right) \oplus b = P(x_{S_i}) \mid x_{S_i} \in H\right] > \frac{1}{2} + \frac{\varepsilon}{\delta m}
$$

Using an averaging argument we can fix $r_{i+1},\ldots,r_m$, b, and all the bits of x outside S_i while preserving the prediction advantage. Renaming x_{S_i} as z, we now observe that z varies uniformly over H while $P(x_{S_j})$ for

$j < i$ and $Q(x_{S_j}, r_j)$ for $j > i$ are now functions P_j of z that depend on only $|S_i \cap S_j| \le \log m$ bits of z. So, we have

$$\Pr_z \left[C\left(P_1(z) \cdots P_{i-1}(z) b P_{i+1}(z) \cdots P_m(z) \right) \oplus b = P(z) \right] > \frac{1}{2} + \frac{\varepsilon}{\delta m}.$$

Each P_j can be computed by a circuit of size $O(m)$, since every function of $\log m$ bits can be computed by a circuit of that size (see, e.g., [Weg87, Ch. 4]). Incorporating these circuits and b into C, we obtain a circuit C' of size size$(C) + O(m^2)$ such that $\Pr_z[C'(z) = P(z)] > \frac{1}{2} + \frac{\varepsilon}{\delta m}$.

Now, since H is $(\varepsilon/\delta m)$-hardcore for P as in Theorem 8, C' must have size greater than $\Omega(\delta^2 \cdot (\varepsilon^2/\delta m)^2) \cdot s = \Omega(\varepsilon^2/m^2) \cdot s$, and hence C must have size greater than $\Omega(\varepsilon^2/m^2) \cdot s - O(m^2)$. ■

Thus, using a mildly hard predicate with the NW generator, we can obtain many bits of crude pseudorandomness. A natural next step would be to try to "extract" this crude pseudorandomness and obtain an output that is indistinguishable from the uniform distribution. Unfortunately, one cannot hope to extract uniformly distributed bits from a distribution that just has high Shannon entropy. Extraction is only possible from distributions that have high *min-entropy*. In the next section, we show how a small modification to the construction achieves what we need.

3.3 A pseudoentropy generator

The reason that we were only able to argue about Shannon entropy in Theorem 9 is that we could only say that δm x_i's land in H *on average*. To obtain a result about min-entropy, we would need to guarantee that many x_i's lie in H *with high probability*. This would be the case if the x_i's were generated pairwise independently instead of via the NW generator. But we also need the special properties of the NW generator to make the current argument about indistinguishability work. We resolve this dilemma by taking the XOR of the two generators to obtain a new generator with the randomness properties of each, similar to the way Impagliazzo and Wigderson [IW97] take the XOR of the NW generator with a random walk on an expander. That is, we obtain $x_1, \ldots, x_m$ from a seed x using the NW generator, we obtain $y_1, \ldots, y_m$ pairwise independent from a seed y, and then use $z_1 = x_1 \oplus y_1, \ldots, z_m = x_m \oplus y_m$ as the inputs to the predicate P. As we will prove shortly, this gives a generator whose output is indistinguishable from some distribution with high min-entropy, as desired.

Recall that we need a way of generating many pairwise independent strings from a short seed.

Lemma 10 ([CG89] (see also [Gol97])) *For any $\ell \in \mathbb{N}$ and $m \le 2^\ell$, there is a generator $\mathrm{PI}_{\ell,m} \colon \{0,1\}^{3\ell} \to (\{0,1\}^\ell)^m$ such that for y selected uniformly at random, the random variables $\mathrm{PI}_{\ell,m}(y)_1, \ldots, \mathrm{PI}_{\ell,m}(y)_m$ are pairwise independent. Moreover $\mathrm{PI}_{\ell,m}$ is computable in time $\mathrm{poly}(\ell, m)$.*

Let $P \colon \{0,1\}^\ell \to \{0,1\}$ be any predicate, let m be any positive integer, and let d be the seed length of $\mathrm{NW}_{\ell,m}$. Then our pseudoentropy generator using P is a function $\mathrm{PE}^P_{\ell,m} \colon \{0,1\}^{d+3\ell} \to \{0,1\}^m$ given by

$$\mathrm{PE}^P_{\ell,m}(x,y) = P(x_1 \oplus y_1) P(x_2 \oplus y_2) \cdots P(x_m \oplus y_m),$$

where

$$(x_1, \ldots, x_m) = \mathrm{NW}_{\ell,m}(x) \quad \text{and} \quad (y_1, \ldots, y_m) = \mathrm{PI}_{\ell,m}(y)$$

The following theorem confirms that this construction does in fact yield a pseudoentropy generator.

Theorem 11 *Suppose no circuit of size s can compute $P \colon \{0,1\}^\ell \to \{0,1\}$ on more than a $1 - \delta$ fraction of the inputs in $\{0,1\}^\ell$. Then, for any $m \le 2^\ell$, $\mathrm{PE}^P_{\ell,m} \colon \{0,1\}^{d+3\ell} \to \{0,1\}^m$ is a (k, s', ε) pseudoentropy generator, with*

$$\begin{aligned} \textit{seed length} = d + 3\ell &= O(\ell^2/\log m) \\ \textit{pseudoentropy} = k &= \delta m/2 \\ \textit{adversary size} = s' &= \Omega(1/\delta^2 m^4) \cdot s - O(m^2) \\ \textit{adversary's maximum advantage} = \varepsilon &= O(1/\delta m) \end{aligned}$$

Moreover, $\mathrm{PE}_{\ell,m}^P$ *is computable in time* $\mathrm{poly}(m, 2^{\ell^2/\log m})$ *with* m *oracle calls to* P*.*

Proof: Let $\varepsilon_1 = 1/\delta m$. Let H be a $(\varepsilon_1/\delta m)$-hardcore set for P, as given by Theorem 8. Like in the proof of Theorem 9, we consider the following distribution D'.

Distribution D': Choose x uniformly from $\{0,1\}^d$ and y uniformly from $\{0,1\}^{3\ell}$. Let $(x_1, \ldots, x_m) = \mathrm{NW}_{\ell,m}(x)$ and $(y_1, \ldots, y_m) = \mathrm{PI}_{\ell,m}(y)$. If $x_i \oplus y_i \in H$, select $b_i \in \{0,1\}$ uniformly at random; and if $x_i \oplus y_i \notin H$, let $b_i = P(x_i \oplus y_i)$. Output $b_1 \cdots b_m$.

By an argument as in the proof of Theorem 9, it can be shown that no circuit of size $s' = \Omega(\varepsilon_1^2/m^2) \cdot s - O(m^2) = \Omega(1/\delta^2 m^4) \cdot s - O(m^2)$ can distinguish D' from $\mathrm{PE}_{\ell,m}^P$ with advantage greater than ε_1. The only change needed is that y should be fixed at the same time as $r_{i+1}, \ldots, r_m$, b, and all the bits of x outside S_i, and z should be $x_{S_i} \oplus y_i$ rather than just x_{S_i}.

Next we argue that D' has statistical difference at most $4/\delta m$ from some distribution D with min-entropy at least $\delta m/2$. This will complete the proof with $\varepsilon = \varepsilon_1 + 4/\delta m = O(1/\delta m)$, as the advantage of any circuit in distinguishing D from $\mathrm{PE}_{\ell,m}^P$ is at most its advantage in distinguishing D' from $\mathrm{PE}_{\ell,m}^P$ plus the statistical difference between D and D'.

For any $w_1, \ldots, w_m \in \{0,1\}^\ell$, define $\#_H(w_1, \ldots, w_m)$ to be the number of w_i's that are in H. As in the proof of Theorem 9, each $x_i \oplus y_i$ is (individually) uniformly distributed and therefore lands in H with probability δ. By linearity of expectations, the expectation of $\#_H(\mathrm{NW}_{\ell,m}(x) \oplus \mathrm{PI}_{\ell,m}(y))$ is δm. Now, since $\{y_i\}$ are pairwise independent and independent from x, it follows that $\{x_i \oplus y_i\}$ are also pairwise independent. Thus, by Chebyshev's inequality,

$$\Pr_{x,y}\left[\#_H(\mathrm{NW}_{\ell,m}(x) \oplus \mathrm{PI}_{\ell,m}(y)) < \frac{\delta m}{2}\right] < \frac{\delta m}{(\delta m/2)^2} = \frac{4}{\delta m}.$$

Therefore, D' has statistical difference at most $4/\delta m$ from the following distribution D:

Distribution D: Choose x uniformly from $\{0,1\}^d$ and y uniformly from $\{0,1\}^{3\ell}$. Let $(x_1, \ldots, x_m) = \mathrm{NW}_{\ell,m}(x)$ and $(y_1, \ldots, y_m) = \mathrm{PI}_{\ell,m}(y)$. If $\#_H(x_1 \oplus y_1, \ldots, x_m \oplus y_m) < \delta m/2$, output a uniformly selected string from $\{0,1\}^m$. Otherwise, select $b_1 \cdots b_m$ as in D' and output $b_1 \cdots b_m$. That is, if $x_i \oplus y_i \in H$, select $b_i \in \{0,1\}$ uniformly at random; and if $x_i \oplus y_i \notin H$, let $b_i = P(x_i \oplus y_i)$.

Now we argue that D has min-entropy at least $\delta m/2$. Let v be any string in $\{0,1\}^m$. Then, conditioned on any x and y, the probability that D outputs v is at most $2^{-\delta m/2}$, since in all cases at least $\delta m/2$ of the output bits of D are selected uniformly and independently. Thus, $\Pr[D = v] = \mathrm{E}_{x,y}[\Pr[D|_{x,y} = v]] \leq 2^{-\delta m/2}$, as desired. ■

3.4 Extracting the randomness

Now we argue that composing a pseudoentropy generator with an (efficient) extractor yields a pseudorandom generator. The manner of composition is illustrated in Figure 2.

Lemma 12 *Suppose* $G: \{0,1\}^{d_1} \to \{0,1\}^m$ *is a* (k, s, ε_1) *pseudoentropy generator and* $\mathrm{Ext}: \{0,1\}^m \times \{0,1\}^{d_2} \to \{0,1\}^n$ *is a* (k, ε_2)*-extractor computable by circuits of size* t*. Then* $G': \{0,1\}^{d_1+d_2} \to \{0,1\}^n$ *defined by* $G'(u,v) = \mathrm{Ext}(G(u), v)$ *is a* $(s-t, \varepsilon_1 + \varepsilon_2)$ *pseudorandom generator.*

Proof: Let D be the distribution of min-entropy k that cannot be distinguished from $G(U_{d_1})$. Suppose $C: \{0,1\}^n \to \{0,1\}$ is a circuit of size $s-t$ that distinguishes $G'(U_{d_1}, U_{d_2})$ from uniform with advantage greater

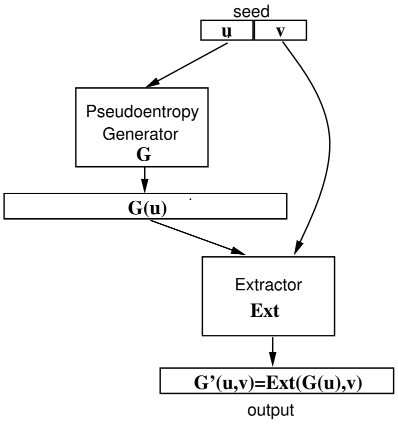

Figure 2: Pseudoentropy Generator + Extractor ⇒ Pseudorandom Generator

than $\varepsilon_1+\varepsilon_2$. By complementing C if necessary, we have $\Pr[C(G'(U_{d_1},U_{d_2}))=1]-\Pr[C(U_n)=1]>\varepsilon_1+\varepsilon_2$. Let $C'\colon\{0,1\}^m\times\{0,1\}^{d_2}\to\{0,1\}$ be the circuit of size s given by $C'(x,v)=C(\textsc{Ext}(x,v))$. Then

$$\begin{aligned}\Pr[C'(G(U_{d_1}),U_{d_2})=1]-\Pr[C'(D,U_{d_2})=1] &= \Pr[C(G'(U_{d_1},U_{d_2}))=1]-\Pr[C(\textsc{Ext}(D,U_{d_2}))=1]\\ &\geq \Pr[C(G'(U_{d_1},U_{d_2}))=1]-\Pr[C(U_n)=1]-\varepsilon_2\\ &> \varepsilon_1,\end{aligned}$$

where the second-to-last inequality follows from the fact that $\textsc{Ext}(D,U_{d_2})$ and U_n have statistical difference at most ε_2. Now, by an averaging argument, the second argument of C' can be fixed to some $v\in\{0,1\}^{d_2}$ to obtain a circuit $C''(x)=C'(x,v)$ of size at most s which distinguishes $G(U_{d_1})$ from D with advantage greater than ε_1. This is a contradiction. ■

Summing up, we have the following theorem:

Theorem 13 *There is a universal constant $\gamma>0$ such that the following holds. Let $P\colon\{0,1\}^\ell\to\{0,1\}$ be any predicate such that no circuit of size s can compute P correctly on more than a $1-\delta$ fraction of the inputs, where $s\leq 2^\ell$ and $\delta\geq s^{-\gamma}$. Define $n=s^\gamma$ and $m=2n^2/\delta$ and let $\mathrm{PE}^P_{\ell,m}\colon\{0,1\}^{d_1}\to\{0,1\}^m$ be the $(\delta m/2,\Omega(1/\delta^2m^4)\cdot s-O(m^2),O(1/\delta m))$ pseudoentropy generator of Theorem 11 and let* $\textsc{Ext}\colon\{0,1\}^m\times\{0,1\}^{d_2}\to\{0,1\}^n$ *be the $(\delta m/2,1/\delta m)$-extractor of Theorem 7. Let* $\text{PE-PRG}^P\colon\{0,1\}^{d_1+d_2}\to\{0,1\}^n$ *be defined by* $\text{PE-PRG}^P(u,v)=\textsc{Ext}(\mathrm{PE}^P_{\ell,m}(u),v)$.

Then, PE-PRG^P *is a (s',ε) pseudorandom generator with*

$$\begin{aligned}\textit{output length}=n &= s^\gamma\\ \textit{seed length}=d_1+d_2 &= O\left(\frac{\ell^2}{\log s}\right)\\ \textit{adversary size}=s' &= \sqrt{s}\\ \textit{adversary's maximum advantage}=\varepsilon &= O(1/n^2),\end{aligned}$$

Moreover, PE-PRG^P *can be evaluated in time $2^{O(\ell^2/\log s)}$ with $O(n^2/\delta)$ oracle calls to P.*

In particular, suppose P is a predicate in E such that no circuit of size $s=2^{\theta\ell}$ can compute P correctly on more than a $1-\delta=1-1/\mathrm{poly}(\ell)$ fraction of the inputs. Then the output length is $n=2^{\Omega(\ell)}$, the seed length is $O(\ell)=O(\log n)$, no circuit of size $s'=2^{\Omega(\ell)}$ can distinguish the output from uniform, and the generator can be evaluated in time $\mathrm{poly}(n)$, so the resulting pseudorandom generator is sufficiently strong to obtain $\mathsf{P}=\mathsf{BPP}$.

Proof: By Theorem 11,

$$d_1 = O\left(\frac{\ell^2}{\log m} + \ell\right) \leq O\left(\frac{\ell^2}{\log s}\right).$$

By Theorem 7,

$$d_2 = O\left(\frac{\log^2\left(\frac{m}{1/\delta m}\right)}{\log(\delta m/2)}\right) = O(\log s) \leq O\left(\frac{\ell^2}{\log s}\right),$$

and Ext is computable by a circuit of size $t = \mathrm{poly}(m, d_2) = \mathrm{poly}(m)$. By Lemma 12, no circuit of size s' can distinguish the output of PE-PRG from uniform with advantage greater than $O(1/\delta m) = O(1/n^2)$, where

$$s' = \Omega(1/\delta^2 m^4) \cdot s - O(m^2) - t \geq \Omega(s^{1-10\gamma}) - \mathrm{poly}(s^\gamma)$$

By choosing γ sufficiently small, s' will always be at least $\sqrt{s}$. ■

Remark 14 As mentioned earlier, Håstad et al. [HILL99] introduced the notion of a pseudoentropy generator and showed that the crude pseudorandomness of such a generator can be extracted to yield a pseudorandom generator. Their work is in the Blum–Micali–Yao setting, in which the generators must be computable in time polynomial in the *seed length* and hence one can only hope for the output to be polynomially longer than the seed (rather than exponentially, as we obtain). Hence throughout their construction they can afford super-linear increases in seed length, whereas preserving the seed length up to linear factors is crucial for obtaining pseudorandom generators good enough for $\mathsf{P} = \mathsf{BPP}$. For example, they can afford to use randomness-inefficient extractors such as 2-universal hash functions, whereas we require extractors which use only a logarithmic number of truly random bits, which have only been constructed recently (first in [Zuc96]). Indeed, the term "extractor" was not even present when the work of [HILL99] first appeared and the first constructions of randomness-efficient extractors used their Leftover Hash Lemma as a starting point.

Remark 15 The output of the pseudoentropy generator $\mathrm{PE}^P_{\ell,m}$ constructed in Theorem 11 is actually "nicer" than stated. Specifically, it is indistinguishable from an *oblivious bit-fixing source* — that is, a distribution on strings of length m in which $m-k$ bit positions are fixed and the other k bit positions vary uniformly and independently. Such sources were the focus of the "bit extraction problem" studied in [Vaz85, BBR85, CGH+85, Fri92] and the term "oblivious bit-fixing source" was introduced in [CW89]. To see that the output of $\mathrm{PE}^P_{\ell,m}$ is indistinguishable from an oblivious bit-fixing source, simply observe that the distribution D given in the proof of Theorem 11 is such a source.[4] Extracting from oblivious bit-fixing sources in which all but k bits are fixed is an easier task than extracting from a general source of min-entropy k, and already in [CW89] there are (implicitly) extractors sufficient for our purposes.

Another point about the output of $\mathrm{PE}^P_{\ell,m}$ is that its *pseudoentropy rate* (i.e., the (pseudo-)min-entropy divided by its length) is at least $\delta/2$, where P is hard to compute correctly on more than a $1-\delta$ fraction of inputs. This means that if P has "constant average-case hardness," it suffices to use a good extractor for constant entropy rate, such as those in [Zuc96, SZ99, Zuc97].

Remark 16 It is natural to ask whether similar ideas can be used to directly construct BMY-type pseudorandom generators from mild hardness. Specifically, consider a modification of the BMY-construction [BM84, Yao82] of pseudorandom generators from strong (i.e., very hard-on-average) one-way permutations, replacing the strong one-way permutation with a weak (i.e., mildly hard-on-average) one. In analogy with Theorem 9, one might hope that the resulting generator has output which is indistinguishable from having high Shannon entropy. Unfortunately, this is not the case in general, at least not to the extent one might expect.

To see this, let us recall the BMY construction. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation, and let $b : \{0,1\}^n \to \{0,1\}$ be a hardcore predicate for f, so no polynomial-time algorithm can predict $b(x)$ from

[4] Actually, D is a *convex combination* of oblivious bit-fixing sources. Distribution X is said to be a convex combination of distributions $X_1, \ldots, X_t$ if there is a distribution I on $\{1, \ldots, t\}$ such that X can be realized by choosing $i \in \{1, \ldots, t\}$ according to I, taking a sample x from X_i, and outputting x. It is easy to see that any extractor for oblivious bit-fixing sources also works for convex combinations of them.

$f(x)$ with inverse-polynomial advantage over the choice of x. Then the generator $G_{f,b} : \{0,1\}^n \to \{0,1\}^k$ is defined by $G_{f,b}(x) = b(x)b(f(x))b(f^2(x)) \cdots b(f^{k-1}(x))$. It is shown in [BM84, Yao82] that, as long as $k = n^{O(1)}$, the output of $G_{f,b}$ cannot be distinguished from uniform by any polynomial-time algorithm.

Now we show how to construct a weak one-way permutation F (and a predicate B so that $B(x)$ is mildly unpredictable from $F(x)$) for which the output of $G_{F,B}$ is distinguishable from *every* distribution of high Shannon entropy. To construct F, let $f : \{0,1\}^n \to \{0,1\}^n$ be a strong one-way permutation with hardcore bit $b : \{0,1\}^n \to \{0,1\}$ as above. Let $t = \lceil \log 2n \rceil$. F will be a permutation on strings of length $n+t$, where the last t bits are viewed as an integer from 0 to $2^t - 1$. For $x \in \{0,1\}^n$ and $i \in \{0,\dots,2^t-1\}$, we define

$$\begin{aligned} F(x,i) &= \begin{cases} (x, i+1 \pmod{2^t}) & \text{if } i \in \{0,\dots,n-1\} \\ (f(x), i+1 \pmod{2^t}) & \text{otherwise.} \end{cases} \\ B(x,i) &= \begin{cases} x_{i+1} & \text{if } i \in \{0,\dots,n-1\} \\ b(x) & \text{otherwise} \end{cases}, \end{aligned}$$

where x_{i+1} denotes the $i+1$'st bit of x. It is easy to verify that no polynomial-time algorithm can invert F on more than, say, $3/4$ of the inputs and similarly $B(x,i)$ cannot be predicted from $F(x,i)$ with probability greater than, say, $7/8$. On the other hand, from the first $2^t + n$ bits of $G_{F,B}(x,i)$, it is easy to predict the remaining bits with probability 1: $2^t + n$ successive applications of F always pass through a sequence of points of the form $(y,0),(y,1),\dots,(y,n-1)$, during which the hardcore bits completely reveal y. All further applications of F and B are then polynomial-time computable given y. Therefore the output of $G_{F,B}$ is distinguishable from any distribution with Shannon entropy greater than $2^t + n = O(n)$, whereas an analogy with Theorem 9 would expect indistinguishability from Shannon entropy $k/8$ (since B cannot be predicted with probability more than $7/8$). The mild hardness of F and B can be varied in this counterexample by increasing or decreasing t relative to $\log n$.

4 List decoding and amplification of hardness

Recall the main theorem of Nisan and Wigderson (Theorem 4) which states that given a sufficiently hard-on-average predicate $P\colon \{0,1\}^\ell \to \{0,1\}$, one can get a pseudorandom generator. To obtain such a predicate, Impagliazzo and Wigderson [IW97] start from a predicate P' that is hard in the worst case (i.e., no small circuit computes it correctly on all inputs) and use a low-degree extension of P' to obtain a multivariate-polynomial function $\hat{p}$ that is mildly hard on the average (as in [BFNW93]). They then apply two different XOR lemmas to obtain functions that grow harder; eventually obtaining as hard a function as required in Theorem 4. We use an alternate approach for this sequence by showing directly that the function $\hat{p}$ above is very hard; as hard as required for Theorem 4. (Strictly speaking, Theorem 4 requires hard *Boolean* functions. This requirement is weakened, both in the original result of [IW97] and implicitly in our result, by using the Goldreich-Levin [GL89] construction of hardcore predicates from hard functions.)

In the process, we discover a connection between amplification of the hardness of functions and efficient decoding of error-correcting codes. In what follows, we describe the decoding properties that we need, why they suffice for amplification of hardness, and how multivariate polynomials yield codes with such decoding properties. For the last part, we use a result of Arora and Sudan [AS97], which involves a technically hard proof. We also provide a simpler proof of their result, with some improved parameters. (These improvements are not needed for the hardness amplification.)

4.1 Notation and Definitions

We will be working with error-correcting codes over arbitrary alphabets. A word or vector over a q-ary alphabet is simply an element of $[q]^n$. It will often be more convenient to think of such a vector as a function mapping $[n]$ to $[q]$. We will switch between these two representations frequently.

Definition 17 *For positive integers n, k, q with $n \geq k$, an $(n,k)_q$* code *$\mathcal{C}$ is an injective map from $[q]^k$ to $[q]^n$. Elements of the domain of $\mathcal{C}$ are referred to as* messages*, and elements of the image are referred to as* codewords.

For codes to be of use to us, we will need that the codewords are sufficiently "far" from each other. So we define the *Hamming distance* between two vectors $x, y \in [q]^n$ to be the number of coordinates i such that $x(i) \neq y(i)$. (Notice we are already using the functional notation!) The *relative Hamming distance*, denoted $\Delta(x, y)$, is $\Pr_{i \in [n]}[x(i) \neq y(i)]$.

In the codes that we construct and use, we will expect that any two codewords are far from each other. But we won't impose such a restriction explicitly. We will rather impose a restriction that the codewords allow for recovery, even after many errors have occurred.

Definition 18 *An $(n, k)_q$ code $\mathcal{C}$ is (ε, l)* list-decodable *if for every word $r \in [q]^n$, there exist at most l codewords $c \in \mathcal{C}$ such that $\Delta(r, c) \leq 1 - (\varepsilon + \frac{1}{q})$. (In other words, at most l codewords agree with any word r in a $(\varepsilon + \frac{1}{q})$-fraction of the coordinates.) r is referred to as the* received word.

Remark 19 Note that the parameter ε is expected to be between 0 and $1 - 1/q$, with a smaller ε indicating a better code (for any fixed choice of the other parameters). Note that even at $\varepsilon = 0$, the fraction of errors allowed is only $1 - 1/q$. This is an information-theoretic upper limit on the fraction of errors one can correct (for any meaningful notion of correction) in a q-ary code, since a random word agrees with any (or even most) codewords in approximately a fraction $1/q$ of the coordinates. Below, we will initially discuss codes in which q is large and $\varepsilon \approx \sqrt{1/q}$ and then use concatenation to obtain $q = 2$ while ε remains close to 0.

Of course, to make these codes useful, we will have some computational requirements. We will need an infinite family of codes, one for every k, capable of encoding k letters of the alphabet into some n letters. These codes should be uniformly constructible, efficiently encodable,and efficiently list-decodable. We will formalize all these notions in the next definition. Two of these aspects, uniform constructibility and efficient encodability are defined along standard lines. However the third aspect, list-decodability, will not be defined along standard lines. We outline the non-standard aspects first:

- First, we will not expect the list-decoding algorithm to return one codeword, but rather a list of up to l codewords such that all nearby codewords are included in the list. This is natural given our definition of (ε, l) list-decodable codes.
- Next, we will expect the list-decoding algorithm to work in time polynomial in $\log k$ and $1/\varepsilon$. This is impossible in a conventional model of computation, since it takes time at least $n \geq k$ to even read the received word. However (and this is where the functional view of words becomes important) we will allow the input and output of the list-decoding algorithm to be specified implicitly. Thus we will assume we have oracle access to the received word r (the input). We will also output the codewords implicitly, by programs that compute the function represented by the codeword. These programs will be allowed to make oracle calls to the received word r. Thus both our decoding algorithm and their output programs are best thought of as oracle-machines. We will use the notation $M^{\mathcal{O}}(x)$ to denote the computation of an oracle-machine M on input x with access to an oracle $\mathcal{O}$. When asking for efficient list-decoding, we will expect that the decoding algorithm, as well as its output programs are all efficient.
- Finally, we will allow our decoding algorithms, as well as their output programs, to be randomized. Below we define what it means for randomized algorithms to approximately compute functions, and to solve search problems.

Definition 20 *A randomized procedure A is said to* compute *a function $f : X \to Y$ at a point $x \in X$ if $\Pr[A(x) = f(x)] \geq 3/4$, where the probability is taken over the internal coin tosses of A. We say that A has* agreement $\alpha \in [0, 1]$ *with f if A computes f on an α fraction of the inputs in X. We say that A* computes *f if it has agreement 1 with f. A randomized procedure A is said to* solve *a search problem S, if on input x, $\Pr[A(x) \in S(x)] \geq 3/4$.*

Remark 21 Often we want the success probability of randomized procedures to be higher than the 3/4 required in the above definition. Although for arbitrary search problems, there are no generic techniques to reduce error, it will always be possible to do so in the cases we are interested. For example, in list decoding, where $S(x)$ is the set of lists which include all nearby codewords, we can amplify by running the list-decoding algorithm several times and outputting the union of the lists. Similarly, for computing functions, a majority vote of several runs can be used. In both cases, the success probability can be increased to $1-\gamma$ with an $O(\log(1/\gamma))$ slowdown in efficiency. As a consequence, in a nonuniform model of computation (e.g. circuits) we can set $\gamma = 1/|X|$ and then fix the coin tosses of A to obtain a *deterministic* procedure solving the same problem with only a $O(\log|X|)$ slowdown in efficiency (as in [Adl78]).

We are now ready to define codes that are "nice" for our purpose. These codes are parameterizd by two parameters: an integer k that counts (roughly) the length of the message to be encoded; and a positive real number ε that is related to the fraction of error from which we expect to be able to recover using list-decoding. Specifically, we expect that a fraction $1-(\varepsilon+\frac{1}{q})$ of errors should be efficiently list-decodable, in a q-ary code.

Definition 22 *A family of codes $\mathcal{C} = \{\mathcal{C}_{k,\varepsilon}\}$ is* nice *if there exist functions $n, q, l : \mathbb{Z}\times\mathbb{R}\to\mathbb{Z}$ and a pair of algorithms* (Encode, Decode) *satisfying the following conditions:*

1. *For every k, ε, $\mathcal{C}_{k,\varepsilon} : [q]^k \to [q]^n$ is an (ε, l)-list-decodable code, where $n = n(k,\varepsilon) \leq \mathrm{poly}(k, 1/\varepsilon)$, $q = q(k,\varepsilon) \leq \mathrm{poly}(k,1/\varepsilon)$ and $l = l(k,\varepsilon) \leq \mathrm{poly}(\log k, 1/\varepsilon)$.*
2. Encode$(x; k, \varepsilon)$ *runs in time* $\mathrm{poly}(n)$ *and returns $\mathcal{C}_{k,\varepsilon}(x)$, where $n = n(k,\varepsilon)$.*
3. Decode$^r(k,\varepsilon)$ *(i.e. with oracle access to a word $r \in [q]^n$) runs in time* $\mathrm{poly}(\log k, 1/\varepsilon)$ *and outputs a list of oracle machines $M_1, \ldots, M_l$ s.t. for every message $x \in [q]^k$ satisfying $\Delta(r, \mathcal{C}_{k,\varepsilon}(x)) \leq 1-(\varepsilon+\frac{1}{q})$, there exists $j \in [l]$ such that M_j^r computes x.* Decode *as well as the M_j's are allowed to be randomized. The running time of M_j is bounded by* $\mathrm{poly}(\log k, 1/\varepsilon)$.

A family of codes is binary *if $q(k,\varepsilon) = 2$.*

Remark 23 Note that the condition $l = l(k,\varepsilon) = \mathrm{poly}(\log k, 1/\varepsilon)$, explicitly specified in Condition 1, is also implicitly enforced by Condition 3 above, since the list-decoding algorithm has to be able to enumerate l machines in time $\mathrm{poly}(\log k, 1/\varepsilon)$. Thus one could safely drop this part of Condition 1 without changing the definition.

4.2 Nice binary codes suffice for amplification of hardness

We first show that nice binary codes suffice to obtain functions that are as hard as required for Theorem 4, given any predicate that is hard in the worst-case.

Theorem 24 *Let $\mathcal{C}$ be a nice family of binary codes. Then there exists a constant c such that the following is true. Let $P : \{0,1\}^\ell \to \{0,1\}$ be a function such that no circuit of size s computes P. Given $\varepsilon > 0$, define $P' : \{0,1\}^{\ell'} \to \{0,1\}$ by $P' = \mathcal{C}_{2^\ell,\varepsilon}(P)$.*[5] *Then no circuit of size $s' = (\varepsilon/\ell)^c \cdot s$ computes the predicate $\mathcal{C}_{2^\ell,\varepsilon}(P)$ correctly on more than a $\frac{1}{2}+\varepsilon$ fraction of the inputs.*

In particular, taking $\varepsilon = 1/s^\gamma$ and assuming $\ell < s^\gamma$ for a sufficiently small constant γ (e.g., $\gamma = 1/4c$), P' has the following parameters:

$$\begin{aligned} \textit{input length} = \ell' &= O(\ell) \\ \textit{adversary size} = s' &= \sqrt{s} \\ \textit{adversary's maximum advantage} = \varepsilon &= 1/s^\gamma, \end{aligned}$$

Moreover, P' can be evaluated in time $2^{O(\ell)}$ with access to the entire truth table of P.

[5] Here we are again viewing messages and codewords as functions. Since the codes are binary, the functions are Boolean.

Proof: Let $k = 2^\ell$. Assume for contradiction that B is a circuit of size $s' = (\varepsilon/\ell)^c \cdot s$ that computes $\mathcal{C}_{2^\ell,\varepsilon}(P)$ correctly on more than a $\frac{1}{2}+\varepsilon$ fraction of the inputs. Then, the decoding algorithm $\textsc{Decode}^B(k,\varepsilon)$ outputs a list of programs $M_1, \dots, M_l$ such that for some j, M_j^B computes P correctly. Since the running times of the algorithms M_j are bounded by a polynomial in $\log k$ and $1/\varepsilon$, we can express M_j^B as a circuit (with some random inputs) of size at most $(\ell/\varepsilon)^{c'}$ for some constant c'. This circuit will involve some oracle calls to B. Throwing in the circuit for B in place of all the oracle calls increases the size of the circuit to at most $(\ell/\varepsilon)^{c'} \cdot s'$.[6] By Remark 21, we can get rid of the randomness at the cost of increasing the circuit size by a factor of $O(\ell') = O(\ell)$ to $(\ell/\varepsilon)^{c'+1} \cdot s'$. Setting $c = c'+1$, we get the desired contradiction. ■

We prove the existence of nice families of binary codes in two steps. First we show that multivariate polynomials lead to a nice family of codes over a growing alphabet. Then we use that to construct a nice family of binary codes.

Lemma 25 *A nice family of codes with* $q(k,\varepsilon) = \mathrm{poly}(\log k, 1/\varepsilon)$, $n(k,\varepsilon) = \mathrm{poly}(k)$, *and* $l(k,\varepsilon) = O(1/\varepsilon)$ *exists.*

Remark 26 The proof will show that the alphabet size $q(k,\varepsilon)$ is at least $1/\varepsilon$. This property will be used later.

Proof: The encoding scheme will interpret the message as the values of a multivariate polynomial on a specified subset of points. The encoding will be the evaluation of the polynomial at all inputs. Below, we specify the choice of the parameters: m, the number of variables, F, the field and H, where H^m is the subset of points where the polynomial is specified by the message.

Given k, ε, we pick a field F of cardinality $(c\log k)^2/\varepsilon^3$ for a constant c to be determined later; and a subset $H \subseteq F$ of cardinality $(\log k)/\varepsilon$ and set $m = (\log k)/(\log |H|)$. We let $q = |F|$ and associate the set $[q]$ with F. Let $b : [k] \to H^m$ be any injective map. To encode a string $x \in F^k$, we find a polynomial $\hat{p} : F^m \to F$ of degree at most $|H| - 1$ in each of the m variables satisfying $\hat{p}(b(i)) = P(i)$ for every $i \in [k]$. (Such a function does exist and can be found easily. The function may be made unique by forcing $\hat{p}(z) = 0$ for all $z \in H^m \setminus \mathrm{image}(b)$.) Letting $n = |F|^m$ and associating $[n]$ with F^m, the encoding of x is simply the polynomial function $\hat{p} : [n] \to F$. Note that, with these settings,

$$\log n = m \cdot \log |F| = \frac{(\log k) \cdot (\log |F|)}{\log |H|} = O(\log k),$$

since $\log |F| = O(\log |H|)$. Thus $n = \mathrm{poly}(k)$, as claimed.

The uniform constructibility and efficient encoding properties are standard. The decoding problem reduces to a "polynomial reconstruction" problem: Given oracle access to a function $f : F^m \to F$, (implicitly) find a list of all total degree d polynomials that agree with f on at least an $\varepsilon + \frac{1}{|F|}$ fraction of the places. Arora and Sudan [AS97] give an efficient solution to this problem. In Theorem 29, we give a simpler algorithm and analysis with improved parameters. In particular, the theorem gives a solution to this problem provided $\varepsilon + 1/|F| \geq c\sqrt{d/|F|}$, for some choice of the constant c. We need only verify that this condition is satisfied for the choice of parameters above. With our choice of parameters,

$$\frac{d}{|F|} < \frac{m \cdot |H|}{|F|} \leq \frac{(\log k) \cdot (\log k)/\varepsilon}{(c\log k)^2/\varepsilon^3} = \frac{\varepsilon^2}{c^2},$$

so the required condition is met. The algorithm of Theorem 29 runs in time $\mathrm{poly}(m, d, \log |F|, 1/\varepsilon) = \mathrm{poly}(\log k, 1/\varepsilon)$, and produces a list of at most $l = O(1/\varepsilon)$ codewords. ■

[6] For simplicity, we have bounded the number of oracle calls by the running time, which in turn we have made a polynomial of unspecified degree in $\log k$ and $1/\varepsilon$. Clearly, to obtain quantitatively better results, one should optimize and compute the number of oracle calls to the received word in the decoding procedure, as this is the only part of the running time which affects the circuit size *multiplicatively*.

To convert the codes constructed above into binary codes, we "concatenate" them with the *Hadamard code*. For a string $z \in \{0,1\}^k$, the *Hadamard encoding of* z is a 2^k-bit string $\mathrm{Had}(z)$ whose positions are indexed by strings $w \in \{0,1\}^k$. The w-th coordinate of the encoding $\mathrm{Had}(z)$ is $\langle w, z\rangle = \sum_{j=1}^{t} w_j z_j (\mathrm{mod}\ 2)$, where $z_j, w_j \in \{0,1\}$ are the coordinates of w and z. Though the Hadamard code is inefficient with respect to the length of codewords, it does have good list-decoding properties. Specifically, we use the following well-known bound (cf., [GRS98, Thm. 18]).

Lemma 27 *For every k, $\mathrm{Had} : \{0,1\}^k \to \{0,1\}^{2^k}$ is a $(\varepsilon, 1/(4\varepsilon^2))$ list-decodable code for all $\varepsilon > 0$.*

Goldreich and Levin [GL89] have given an efficient list-decoding algorithm for the Hadamard code, which runs in time $\mathrm{poly}(k, 1/\varepsilon)$. However, for us, even brute-force exhaustive search running in time $\mathrm{poly}(2^k)$ will suffice. By "concatenating" the codes of Lemma 25 with the Hadamard code and appropriately combining the list-decoding algorithms, we obtain the following.

Lemma 28 *There exists a nice family of binary codes with parameters $n = \mathrm{poly}(k/\varepsilon)$ and $l = \mathrm{poly}(1/\varepsilon)$*

Proof: Let $\mathcal{C}$ be the code as given by Lemma 25. We obtain a nice family of binary codes $\mathcal{C}'$ as follows.

Given k and ε, first set $\delta = \varepsilon^3/4$ and let (n, q, l) be the parameters of the code $\mathcal{C}_{k,\delta}$. (In particular, $q > 1/\delta$). Let $t = \lceil \log_2 q \rceil$, and let $b : [q] \to \{0,1\}^t$ be any injective map; for $z \in [q]$, we will write $\mathrm{Had}(z)$ as shorthand for $\mathrm{Had}(b(z))$. To encode a string $x \in \{0,1\}^k$, we first encode it using $\mathcal{C}_{k,\delta}$ to get $y = \mathcal{C}_{k,\delta}(x) \in [q]^n$. Then we encode each coordinate of y as a 2^t-bit string using the Hadamard code. Thus the concatenated encoding encodes a k-bit vector x as a vector

$$\mathrm{Had}(y(1))\mathrm{Had}(y(2)) \cdots \mathrm{Had}(y(n)), \qquad \text{where } y = \mathcal{C}_{k,\delta}(x) \in [q]^n.$$

Clearly, the encoding is of length $n' = n \cdot 2^t \leq n \cdot (2q) = \mathrm{poly}(k, 1/\varepsilon)$ bits. It is also clear that the encoding for the concatenated code can be computed efficiently.

We now describe its decoding. The decoding proceeds using the usual paradigm for the decoding of concatenated codes. We first decode each symbol of the "inner" code, i.e., the Hadamard code; and then decode the "outer code"; in each case we use the respective decoding algorithm. The details that need to be verified are: (1) We need to specify the decoding algorithm for the Hadamard code. (2) We have to implement the decoding paradigm with input/outputs being implicit. (3) While decoding the inner-code, we don't get unique answers but rather a list of codewords. We need a list-decoding version of the decoding procedure.

Given k, ε, and an oracle for the received word $r : [n] \times [2^t] \to \{0,1\}$, we implement oracles $r'_1, r'_2, \ldots, r'_{1/\varepsilon^2} : [n] \to [q]$ as follows. Given $i \in [n]$, we consider the oracle $r|_i : [2^t] \to \{0,1\}$ given by $r|_i(j) = r(i,j)$. We find a list of all elements $z \in [q]$ such that $\mathrm{Had}(z)$ has agreement at least $1/2 + \varepsilon/2$ with $r|_i$. By Lemma 27, this list has at most $1/\varepsilon^2$ elements. The oracle r'_m, on query i, outputs the m'th element of this list (after sorting them using some canonical order, such as the lexicographic order). We then invoke the list decoding algorithm for $\mathcal{C}_{k,\delta}$ $1/\varepsilon^2$ times, once for each r'_m, and take the union of the lists obtained. Thus, the resulting list is of length at most $l \cdot (1/\varepsilon^2) = \mathrm{poly}(1/\varepsilon)$.

To analyze the correctness of our decoding algorithm, consider a message x such that $\mathcal{C}'_{k,\varepsilon}(x)$ has $\frac{1}{2} + \varepsilon$ agreement with r. Let $y = \mathcal{C}_{k,\delta}(x)$. An application of Markov's inequality yields that for at least $\varepsilon/2$ fraction of the indices $i \in [n]$, $r|_i$ has at least $1/2 + \varepsilon/2$ agreement with $\mathrm{Had}(y(i)$, and therefore $r|_i = r'_m(i)$ for some j. Since there are only $1/\varepsilon^2$ choices for m, it follows by averaging that there exists a m_0 such that $r'_{m_0}(i) = r|_i$ for at least a fraction $(\varepsilon/2) \cdot \varepsilon^2 = \varepsilon^3/2$ of the indices $i \in [n]$. Since $1/q + \delta \leq 2\delta = \varepsilon^3/2$, the list-decoding algorithm for $\mathcal{C}_{k,\delta}$ will produce a list of up to $l = \mathrm{poly}(1/\varepsilon)$ oracles which includes x. ■

Comparison with [IW97]. Theorem 24 and Lemma 28 provide sufficient hardness amplification to immediately apply the Nisan–Wigderson construction (Theorem 4) and obtain the main result of Impagliazzo and Wigderson [IW97]. Specifically, if P is a predicate in E which cannot be computed by circuits of size $s = 2^{\theta\ell}$, then P' given in Theorem 24 will also be in E, and circuits of size $s' = 2^{\Omega(\ell')}$ will not be able to compute P' with advantage more than $\varepsilon = 2^{-\Omega(\ell')}$. Plugging such a predicate P' into Theorem 4 gives a pseudorandom generator whose seed length is logarithmic in its output length and security, and hence implies $\mathsf{P} = \mathsf{BPP}$.

In addition, our construction provides hardness amplification for other settings of parameters that improves over the hardness amplification of [IW97]. Specifically, the input length of P' is only a constant factor more than that of P (i.e., $\ell' = O(\ell)$), regardless of the security s. In contrast, hardness amplification of [IW97] produces a predicate with input length $\Theta(\ell^2/\log s)$, which is $O(\ell)$ only if $s = 2^{\Omega(\ell)}$. Note, however, that our construction does not remove the $\Theta(\ell^2/\log s)$ overhead in seed length incurred when subsequently applying Theorem 4 to obtain a pseudorandom generator. Obtaining a construction of pseudorandom generators from hard predicates which increases seed length by only a constant factor for all values of the security s is still an open problem.[7] Our result demonstrates that it suffices to solve this problem for very hard-on-average predicates. A solution would have significant implications for the construction of extractors, via the connection between extractors and pseudorandom generators recently established by Trevisan [Tre99].

The derandomized XOR lemma of Impagliazzo and Wigderson does have an important advantage over our hardness amplification technique when one starts with a mildly hard predicate rather than a worst-case hard predicate. Specifically, if $P : \{0,1\}^\ell \to \{0,1\}$ cannot be computed by small circuits on more than a $1-\delta$ fraction of inputs, they obtain a hard-on-average predicate P' that is computable in time $\mathrm{poly}(\ell, 1/\delta)$ with oracle access to P. Our construction, on the other hand, does not take advantage of this mild hardness. Instead, we do a "global" encoding of P, just as if P were worst-case hard, to obtain a hard-on-average predicate P' computable in time $\mathrm{poly}(2^\ell)$ with oracle access to P. It would be interesting to see if mild hardness could be amplified "locally" as in [IW97] using techniques based on error-correcting codes.

4.3 List decoding of multivariate polynomials

Recall that we wish to solve the following problem:
Given: An oracle $f\colon F^m \to F$ and parameters $d \in \mathbb{N}$ and $\varepsilon \in \mathbb{R}$.
Goal: Reconstruct (an implicit representation for) every polynomial of total degree at most d that has ε-agreement with the function f. Specifically, construct randomized oracle machines $M_1, \ldots, M_l$ such that for every polynomial $p\colon F^m \to F$ of degree d that has (relative) agreement ε with f, there exists $j \in [l]$ such that M_j^f computes p.

We will be interested in the running time of the "reconstruction procedure", i.e., the time taken to generate the machines $M_1, \ldots, M_l$, as well as the running times of the machines $M_1, \ldots, M_l$.

Theorem 29 *There exists a constant c such that the reconstruction problem above can be solved in time $\mathrm{poly}(m, d, \log|F|, 1/\varepsilon)$, provided $\varepsilon > c\sqrt{d/|F|}$. Furthermore, the reconstruction algorithm produces a list of at most $l = O(1/\varepsilon)$ oracle machines, each with running time at most $\mathrm{poly}(m, d, \log|F|, 1/\varepsilon)$.*

Remark 30

1. This theorem is a strengthening of a theorem due to [AS97]. In particular, the lower bound on ε here is smaller than that of [AS97], who obtain an unspecified polynomial in d and $1/|F|$. Furthermore, our proof is simpler and in particular does not require "low-degree testing."
2. The bound of $\Omega(\sqrt{d/|F|})$ is within a constant factor of the bound for the univariate case. The constant c above is not optimized in this paper. But our methods can push it down to any constant greater than $\sqrt{2}$ (assuming $d/|F|$ is sufficiently small). For the univariate case, this constant is 1 [GS99]. No inherent reason is known for the gap.

Before proving Theorem 29, we recall that polynomials are sufficiently list-decodable from a combinatorial standpoint (i.e., efficiency considerations aside).

Theorem 31 (cf., [GRS98, Thm. 17]) *For any $f : F^m \to F$ and $\varepsilon \geq \sqrt{2d/|F|}$, the number of total degree d polynomials that have (relative) agreement at least ε with f is less than $2/\varepsilon$.*

[7] We note that this problem has recently been solved by Impagliazzo, Shaltiel, and Wigderson [ISW00] (though their pseudorandom generators have slightly suboptimal output length).

Now we proceed with the proof of Theorem 29. Fix an oracle $f: F^m \to F$ and a degree d polynomial $p: F^m \to F$ with ε agreement with f. We observe that it suffices to reconstruct a (randomized) oracle machine M such that M^f has sufficiently high agreement with p. This is due to the existence of "self-correctors" of polynomials [BF90, Lip89, GLR+91, GS92]. Specifically, we use the following theorem:

Theorem 32 ([GLR+91]) *There exists a randomized oracle machine* Corr *taking as parameters integers d and m and a field F such that on access to a randomized oracle $M: F^m \to F$ with agreement $\frac{15}{16}$ with some degree d polynomial p,* Corr^M *computes p in time* $\mathrm{poly}(d, m)$ *provided* $|F| \geq 2(d+1)$.

As in the algorithms of [BF90, Lip89, GLR+91], we use the properties of "lines" in the m-dimensional space F^m, defined below.

Definition 33 *The* line through $x, y \in F^m$, *denoted $l_{x,y}$, is the parameterized set of points $\{l_{x,y}(t) \stackrel{\text{def}}{=} (1-t)x + ty \mid t \in F\}$. Given a function $f: F^m \to F$, f restricted to the line $l_{x,y}$ is the function $f|_{l_{x,y}}: F \to F$ given by $f|_{l_{x,y}}(t) = f(l_{x,y}(t))$.*

Notice that if f is a polynomial of total degree d, then $f|_{l_{x,y}}(t)$ is a univariate polynomial of degree at most d. Our strategy, to reconstruct the value of p at a random point x, is to look at a random line going through x. On this line, p turns into a univariate polynomial. Furthermore, the random line through the randomly chosen point x is a "pairwise independent" collection of points from F^m. Thus p and f will have agreement close to ε on this line as well. Thus the goal of finding $p(x)$ "reduces" to the goal of reconstructing p restricted to this line, i.e., a univariate reconstruction problem, a problem that has been addressed in [ALRS99, Sud97, GS99]. In particular, we use the following theorem.

Theorem 34 ([Sud97]) *Given a sequence of n distinct pairs $\{(t_i, v_i)\}_{i=1}^n$, $t_i, v_i \in F$ and integer parameters d, k, a list of all polynomials $g_1, \ldots, g_l$ of degree at most d satisfying $|\{i \in \{1, \ldots, n\} | g_j(t_i) = v_i\}| \geq k$, can be reconstructed in time* $\mathrm{poly}(n, \log |F|)$ *provided $k > \sqrt{2dn}$.*

Using the above theorem and the idea of restricting one's attention to random lines, it easy to design an algorithm that, on input x, enumerates a small list of values that includes $p(x)$ (for most x). However, we need to figure out which one of these values is $p(x)$. More specifically, we need to "collate" these values and assign them to a small collection of oracle machines so one of them consistently outputs $p(x)$. To do so, the oracle machine needs some additional information about the polynomial p. (Note that so far, the only information about p that is known is that it is a low-degree polynomial with ε-agreement with f — but there may be many, at least $\Omega(1/\varepsilon)$, such polynomials.) We will show that it suffices to know the value of p at a single (random) point z. Let $p(z) = a$; we design an oracle machine $M_{z,a}$ which now tries to reconstruct the polynomial p. The machine takes as parameters a positive real number ε, integers d and m, and a field F.

- $M_{z,a}(x)$:
  1. (Explicitly) find a list of distinct (univariate) polynomials $g_1, \ldots, g_l$ such that this list includes all polynomials of degree at most d that have agreement at least $\varepsilon/2$ with $f|_{l_{z,x}}$ and does not include any polynomial with agreement less than $\varepsilon/4$.
  2. If there exists a unique index $i \in \{1, \ldots, l\}$ such that $g_i(0) = a$, then output $g_i(1)$, else output anything.

Remark 35 1. Step 1 above can be computed in time polynomial in $1/\varepsilon$, $\log |F|$, m, and d as follows: If F is small enough, then we let $t_1, \ldots, t_n$ be all the elements of F and invoke Theorem 34 on the set $\{(t_i, f(l_{z,x}(t_i)))\}_{i=1}^n$ with $k = \varepsilon n/2$. (Note that $k > \sqrt{2dn}$ as long as $\varepsilon > 2\sqrt{2}\sqrt{d/|F|}$, which is true by hypothesis.) If F is too large to do this, then set $n = \mathrm{poly}(d/\varepsilon)$ and pick $t_1, \ldots, t_n$ distinct points at random from F and then invoke Theorem 34 on the set $\{(t_i, f(l_{z,x}(t_i)))\}_{i=1}^n$ with $k = \varepsilon n/4$. Since there are at most $4/\varepsilon$ polynomials with agreement at least $\varepsilon/2$ with $f|_{l_{z,x}}$ (by Theorem 31), the choice of n guarantees that with high probability, all of these polynomials agree with $f|_{l_{z,x}}$ on at least $\varepsilon n/4$ of the t_i's. As the choice of n also guarantees that $k = (\varepsilon n/4) > \sqrt{2dn}$, Theorem 34 yields a list containing all polynomials with agreement at least $\varepsilon/2$. Now, we wish to discard all polynomials with agreement less than $\varepsilon/4$ — this can be accomplished by comparing each polynomial g obtained with $f|_{l_{z,x}}$ on a random sample of poly$(1/\varepsilon)$ points from F and discarding it if it has agreement smaller than $\varepsilon/3$ on this sample.

2. By Theorem 31, the number of polynomials output in Step 1 above is at most $8/\varepsilon$.

To shed some light on the steps above, consider the actions of the machine $M_{z,a=p(z)}$: We expect that $p|_{l_{z,x}}$ is one of the g_i's returned in Step 1 above. In Step 2 we try to find out which g_i to use by checking to see if there is a unique one which has $g_i(0) = a$ (recall that $p|_{l_{z,x}}(0) = p(z) = a$), and if so we use this polynomial to output $p(x) = p|_{l_{z,x}}(1) = g_i(1)$. This intuition is made precise in Section 4.4. We now finish the description of the reconstruction procedure.

- **Reconstruction algorithm**

  Repeat the following $O(\log(1/\varepsilon))$ times:
  1. Pick $z \in F^m$ at random.
  2. Pick $y \in F^m$ at random.
  3. Find a list of univariate polynomials $h_1, \ldots, h_l$ including all polynomials of degree at most d with agreement at least $\varepsilon/2$ with $f|_{l_{z,y}}$.[8]
  4. For every polynomial h_j, include the oracle machine $\mathsf{Corr}^{M_{z,h_j(0)}}$ in the output list.

4.4 Analysis of the polynomial reconstruction procedure

Now we show that the reconstruction algorithm runs in time poly$(\frac{md}{\varepsilon}\log|F|)$ and outputs a list of oracle machines that includes one for every polynomial p that has ε agreement with f. Theorem 29 follows immediately.

The claim about the running time is easily verified. To analyze the correctness, it suffices to show that in any iteration of Steps 1–4 in **Reconstruction Algorithm**, an oracle machine computing p is part of the output with, say, constant probability for any fixed polynomial p of degree d that has ε agreement with f. We show this in two parts. First we argue that for most choices of z, $M_{z,p(z)}$ is an oracle machine that computes p on 15/16 of all inputs (and thus $\mathsf{Corr}^{M_{z,p(z)}}$ computes p everywhere). Then we show that for most pairs (z, y), there exists j s.t. the polynomial h_j reconstructed in Step 3 satisfies $h_j(0) = p(z)$.

Lemma 36 *There exists a constant c s.t. for every d, F, ε satisfying $1 \geq \varepsilon \geq c\sqrt{d/|F|}$, it is the case that*

$$\Pr_x \left[M_{z,p(z)}(x) = p(x)\right] \geq 15/16,$$

with probability at least $1/2$ over the random choice of $z \in F^m$,

Proof: We first argue that when both x and z are picked at random, certain bad events are unlikely to happen. The next two claims describe these bad events and upper bound their probability.

Claim 37 *If $\varepsilon \geq 256/|F|$, then*

$$\Pr_{x,z} \left[\nexists i \in [l] \text{ s.t. } g_i = p|_{l_{z,x}}\right] \leq 1/64.$$

Proof: For the polynomial $p|_{l_{z,x}}$ not to be included in the output list it has to be the case that p and f do not have $\varepsilon/2$ agreement on the line $l_{z,x}$. But the line is a pairwise independent collection of $|F|$ points in F^m. The quantity of interest then is the probability that a random variable with expectation ε attains an average of at most $\varepsilon/2$ on $|F|$ samples. Using Chebychev's inequality, this probability may be bounded by $\frac{4}{\varepsilon|F|} \leq \frac{1}{64}$. ■

[8] This is done as in Remark 35, though here we do not care if the list contains extra polynomials with low agreement.

Claim 38 *If* $\varepsilon \geq \max\left\{4\sqrt{2d/|F|}, 512d/|F|\right\}$, *then*

$$\Pr_{x,z}\left[\exists j \in [l] \text{ s.t. } g_j \neq p|_{l_{z,x}} \text{ and } p|_{l_{z,x}}(0) = g_j(0)\right] \leq \frac{ld}{|F|} \leq 1/64.$$

Proof: For convenience in this argument, assume that $M_{z,p(z)}$ finds *all* polynomials of agreement at least $\varepsilon/4$ with $f|_{l_{z,x}}$ rather than just a subset, as that is clearly the worst case for the claim.

The lemma would be obvious if the event in consideration had been considering a randomly chosen point on the line $l_{z,x}$, rather than the point $l_{z,x}(0)$. For any $g_i \neq p|_{l_{z,x}}$), the probability $g_i(t) = p|_{l_{z,x}}(t)$ is at most $d/|F|$ for a randomly chosen $t \in F$, so the probability that there exists an i such that $g_i \neq p|_{l_{z,x}}$ and $g_i(t) = p|_{l_{z,x}}(t)$ is at most $l \cdot d/|F|$.

The only complication is that seemingly t is not being chosen at random, but rather set to a fixed value $t = 0$. However, this does not really change the analysis. The polynomials $g_1, \ldots, g_l, p|_{l_{z,x}}$ are only functions of f, p, and the set $L = \{l_{z,x}(t) | t \in F\}$, rather than on the specific parameterization of the line. So one could really think of L being picked first, and then the points x and z being chosen later to be two random points in L. As argued above, the probability (over $z \in L$) that any of the polynomials $g_1, \ldots, g_l$ agree with $p|_{l_{z,x}}$ at the point $l_{z,x}(0) = z$ is at most $ld/||F|$. Since $\varepsilon/4 \geq \sqrt{2d/|F|}$, Theorem 31 gives $l < 8/\varepsilon = |F|/(128d)$ and the claim follows. ■

Discounting for the two possible bad events considered in Claims 37 and 38, we find that with probability at least $1 - 1/32$, there exists a polynomial g_i returned in Step 1 of $M_{z,p(z)}$ such that $g_i = p|_{l_{z,x}}$; furthermore, this is the unique polynomial such that $g_i(0) = p|_{l_{z,x}}(0) = p(z)$. Thus the output is $g_i(1) = p|_{l_{z,x}}(1) = p(x)$. Thus with probability at least $31/32$, we find that for a random pair (z, x), $M_{z,p(z)}$ computes $p(x)$. An application of Markov's inequality now yields the desired result. ■

Lemma 39 *With probability at least* $1 - 1/64$, *one of the polynomials reconstructed in any one execution of Step 3 of* **Reconstruction Algorithm** *is* $p|_{l_{z,y}}$; *and thus one of the oracle machines created in Step 4 is* $\mathsf{Corr}^{M_{z,p(z)}}$, *provided* $\varepsilon > 256/|F|$.

Proof: As in Claim 37 we argue that p and f have at least $\varepsilon/2$ agreement on the line $l_{z,y}$ and then $p|_{l_{z,y}}$ is one of the polynomials output in this step. Thus one of the oracle machines created is $\mathsf{Corr}^{M_{z,a}}$ for $a = p|_{l_{z,y}}(0) = p(z)$. ■

Proof of Theorem 29: Fix any degree d polynomial p with ε agreement with f. Combining Lemmas 36 and 39 we find that with probability $31/64$, one of the oracle machines output by the reconstruction algorithm is $\mathsf{Corr}^{M_{z,p(z)}}$; and z is such that $M_{z,p(z)}$ computes $p(x)$ for at least $15/16$ fraction of x's in F^m; and thus (by Theorem 32) $\mathsf{Corr}^{M_{z,p(z)}}$ computes p on every input.

By Theorem 31, there are only $O(1/\varepsilon)$ polynomials having at least ε agreement with f. Thus, repeating the loop $O(\log \frac{1}{\varepsilon})$ times ensures that every such polynomial p is included in the output with high probability. The list can be trimmed to size $O(1/\varepsilon)$ by discarding all polynomials having agreement less than $\varepsilon/2$ with f (which can be identified via sampling). ■

Acknowledgments

We thank Oded Goldreich, Venkatesan Guruswami, Peter Bro Miltersen, Amnon Ta-Shma, and Avi Wigderson for clarifying discussions and pointing us to some related work. We also thank the anonymous referees for numerous helpful suggestions.

References

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Ann Arbor, Michigan, 16–18 October 1978. IEEE.

[ACR97] Alexander Andreev, Andrea Clementi, and José Rolim. Worst-case hardness suffices for derandomization: a new method for hardness-randomness trade-offs. In *Proceedings of ICALP'97*, pages 177–187. LNC 1256S, Springer-Verlag, 1997.

[ALRS99] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):487–510, 1999.

[AS97] Sanjeev Arora and Madhu Sudan. Improved low degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4–6 May 1997.

[AK97] V. Arvind and J. Köbler. On resource-bounded measure and pseudorandomness. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249. LNCS 1346, Springer-Verlag, 1997.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48, Rouen, France, 22–24 February 1990. Springer.

[BBR85] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. How to reduce your enemy's information (extended abstract). In Hugh C. Williams, editor, *Advances in Cryptology—CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 468–476. Springer-Verlag, 1986, 18–22 August 1985.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[CPS99] Jin-Yi Cai, A. Pavan, and D. Sivakumar. On the hardness of the permanent. In *16th International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, pages 90–99, Trier, Germany, March 4–6 1999. Springer-Verlag.

[CG89] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, March 1989.

[CGH+85] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem or t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science*, pages 396–407, Portland, Oregon, 21–23 October 1985. IEEE.

[CW89] Aviad Cohen and Avi Wigderson. Dispersers, deterministic amplification, and weak random sources (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 14–19, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. John Wiley & Sons, Inc., 2nd edition, 1991.

[FL96] Uriel Feige and Carsten Lund. On the hardness of computing the permanent of random matrices. *Computational Complexity*, 6(2):101–132, 1996.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22(5):994–1005, October 1993.

[Fri92] Joel Friedman. On the bit extraction problem. In *33rd Annual Symposium on Foundations of Computer Science*, pages 314–319, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[GLR+91] Peter Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 32–42, New Orleans, Louisiana, 6–8 May 1991.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43(4):169–174, 28 September 1992.

[Gol95] Oded Goldreich. *Foundations of Cryptography (Fragments of a Book)*. Weizmann Institute of Science, 1995. Available, along with revised version 1/98, from `http://www.wisdom.weizmann.ac.il/~oded`.

[Gol97] Oded Goldreich. A computational perspective on sampling (survey). Available from `http://www.wisdom.weizmann.ac.il/~oded/`, May 1997.

[Gol99] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness*. Number 17 in Algorithms and Combinatorics. Springer-Verlag, 1999.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GL89] Oded Goldreich and Leonid 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, Seattle, Washington, 15–17 May 1989.

[GNW95] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR lemma. Technical Report TR95–050, Electronic Colloquium on Computational Complexity, March 1995. `http://www.eccc.uni-trier.de/eccc`.

[GRS98] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries — the highly noisy case. Technical Report TR98-060, Electronic Colloquium on Computational Complexity, 1998.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Trans. Inform. Theory*, 45(6):1757–1767, 1999.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396 (electronic), 1999.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 October 1995. IEEE.

[ISW00] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Extractors and pseudorandom generators with optimal seed length. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, Portland, Oregon, 21–23 May 2000.

[IW97] Russell Impagliazzo and Avi 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, El Paso, Texas, 4–6 May 1997.

[IW98] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *36th Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, November 8–11 1998. IEEE.

[KvM99] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 659–667, Atlanta, Georgia, 1–4 May 1999.

[KS98] S. Ravi Kumar and D. Sivakumar. Personal communication, October 1998.

[Lip89] Richard Lipton. New directions in testing. In *Proceedings of DIMACS Workshop on Distributed Computing and Cryptography*, 1989.

[NT99] Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):148–173, 1999.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, February 1996.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, August 1997.

[SZ99] Aravind Srinivasan and David Zuckerman. Computing with very weak random sources. *SIAM J. Comput.*, 28(4):1433–1459 (electronic), 1999.

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

[STV98] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. Technical Report TR98-074, Electronic Colloquium on Computational Complexity, December 1998. `http://www.eccc.uni-trier.de/eccc`.

[STV99a] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma [abstract]. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, page 4, Atlanta, GA, May 1999.

[STV99b] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma [extended abstract]. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 537–546, Atlanta, Georgia, 1–4 May 1999.

[Tre99] Luca Trevisan. Constructions of near-optimal extractors using pseudo-random generators. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 141–148, Atlanta, Georgia, 1–4 May 1999.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Vaz85] Umesh V. Vazirani. Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 366–378, Providence, Rhode Island, 6–8 May 1985.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions*. Wiley, 1987.

[Wig98] Avi Wigderson. Personal communication, October 1998.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

[Zuc96] David Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4/5):367–391, October/November 1996.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.