List-Decoding Using The XOR Lemma

Luca Trevisan*
Computer Science Division, U.C. Berkeley
luca@cs.berkeley.edu

Abstract

We show that Yao's XOR Lemma, and its essentially equivalent rephrasing as a Direct Product Lemma*, can be re-interpreted as a way of obtaining error-correcting codes with good list-decoding algorithms from error-correcting codes having weak unique-decoding algorithms. To get codes with good rate and efficient list decoding algorithms one needs a proof of the Direct Product Lemma that, respectively, is strongly derandomized, and uses very small advice.*

We show how to reduce advice in Impagliazzo's proof of the Direct Product Lemma for pairwise independent inputs, which leads to error-correcting codes with $O(n^2)$ encoding length, $\tilde{O}(n^2)$ encoding time, and probabilistic $\tilde{O}(n)$ list-decoding time. (Note that the decoding time is sub-linear in the length of the encoding.)

Back to complexity theory, our advice-efficient proof of Impagliazzo's "hard-core set" results yields a (weak) uniform version of O'Donnell results on amplification of hardness in NP. We show that if there is a problem in NP that cannot be solved by BPP algorithms on more than a $1 - 1/(\log n)^c$ fraction of inputs, then there is a problem in NP that cannot be solved by BPP algorithms on more than a $3/4 + 1/(\log n)^c$ fraction of inputs, where $c > 0$ is an absolute constant.

1. Introduction

Yao's *XOR Lemma* states that if $f : \{0,1\}^n \to \{0,1\}$ is a boolean function that is hard to compute on more than a $(1-\delta)$ fraction of inputs, then computing $f(x_1) \oplus \cdots \oplus f(x_k)$ on more than a $1/2+\varepsilon$ fraction of the k-tuples $(x_1, \ldots, x_k)$ is also hard, where ε is roughly $(1-\delta)^k$. An essentially equivalent *direct product* lemma states that computing the vector $f(x_1), \cdots, f(x_k)$ for more than an ε fraction of the $(x_1, \ldots, x_k)$ is hard, where again ε is roughly $(1-\delta)^k$. At least four proofs of this result are know [12, 10, 6, 11].

In this paper we show that any black-box proof of the XOR Lemma or of the Direct Product Lemma gives a way to derive error-correcting codes with strong list-decoding algorithms from error-correcting codes with weak unique-decoding algorithms. Applying this idea directly to the standard form of the Direct Product Lemma would give codes with very large encoding length and with decoding algorithms producing very long lists.

As we explain below, the encoding length can be reduced by using a *derandomized Direct Product Lemma*. By this, we mean a Lemma that says that there is a pseudorandom distribution D of k-tuples $(x_1, \ldots, x_k)$ such that D can be sampled using fewer than nk random bits, and such that if $f()$ is hard to compute on more than a $1-\delta$ fraction of inputs, then $(f(x_1), \ldots, f(x_k))$ is hard to compute more than an ε fraction of the times when $(x_1, \ldots, x_k)$ is sampled from D. Two derandomized proofs of the Direct Product Lemma are known. One, by Impagliazzo [10], works if the x_i are pairwise independent. Then only $2n$ random bits are needed to sample a k-tuple. On the other hand, the proof only works for ε equal to about $1/(\delta k)$. Another proof, by Impagliazzo and Wigderson [11], allows ε to be exponentially small in k, and uses $O(n)$ random bits to sample the k-tuple.

Regarding the length of the list, it depends on the amount of *advice* used in the reduction that proves the Direct Product Lemma. The proofs of Levin [12] and Goldreich et al. [6] work even with a relatively small advice, but they require the inputs to be independent and, as mentioned above, this translates to codes with very long encoding length. The derandomized proofs of Impagliazzo [10] and Impagliazzo and Wigderson [11], on the other hand, use substantially large advice.

The main technical contribution of this paper is an advice-efficient version of Impagliazzo's Direct Product Lemma for pairwise independent inputs. Our proof gives a way to convert, say, the Sipser-Spielman codes into codes over large alphabet with quadratic encoding length and with list-decoding algorithms that can handle a fraction of errors arbitrarily close to one, and produce a list of size depending only on the error.

As a main intermediate step, we prove an advice-efficient

* Supported by a Sloan Research Fellowship and an Okawa Foundation Grant.

version of Impagliazzo's result about hard-core sets for weakly hard-on-average functions. O'Donnell [14] used Impagliazzo's hard core set construction to prove a result about amplification of average-case hardness for problems in NP, in the non-uniform setting. We use our result to prove a uniform version of O'Donnell's results.

1.1. Direct Product Lemma and List Decoding

Known proofs of the direct product lemma tend have more or less the following form:

> For $\delta > 0$, integer k, and sufficiently large ε (typically ε is at least some constant times $(1-\delta)^k$), let $f : [N] \to \{0,1\}$ be a function, define $F(x_1, \ldots, x_k) = (f(x_1), \ldots, f(x_k))$ and let G be a function that agrees with F on at least an ε fraction of inputs.
>
> Then there is an oracle circuit A of size $\text{poly}(\log N, 1/\varepsilon, 1/\delta, k)$ that has at most $\text{poly}(1/\varepsilon, 1/\delta)$ oracle gates and such that $\mathbf{Pr}[A^G(x) = f(x)] \geq 1 - \delta$.

Derandomized proofs have a somewhat different form that we will discuss shortly.

The application to coding theory is as follows. Let $C : \mathcal{M} \to \{0,1\}^N$ be an error-correcting code having an efficient decoding algorithm that can correct up to δN errors. For a message $M \in \mathcal{M}$ and an index $x \in [N]$, we denote by $C(M)[x]$ the x-th bit of the encoding of M.

We define a new code $C' : \mathcal{M} \to (\{0,1\}^k)^{N^k}$ with codewords of length N^k over the alphabet $\{0,1\}^k$. Let us identify indices of entries of C' with k-tuples of elements from $[N]$; then C' is defined as follows: for a message M,

$$C'(M)[x_1, \ldots, x_k] = (C(M)[x_1] \cdots C(M)[x_k])$$

That is, $C'(M)$ has an entry for every k entries of $C(M)$, and the entry of $C'(M)$ contains a concatenation of the k bits present in the corresponding k entries of $C(M)$.

Note that if we think of $C(M)$ as a function $f : [N] \to \{0,1\}$, then $C'(M)$ is the function $F : [N]^k \to \{0,1\}^k$ defined in the direct product lemma.

Let now $G \in (\{0,1\}^k)^{N^k}$ be a string having agreement ε with $C'(M)$. From the direct product lemma it follows that there is an oracle circuit A of size $\text{poly}(\log N, 1/\varepsilon, 1/\delta, k)$ such that A^G has agreement $1 - \delta$ with $C(M)$. Given G, we can then enumerate all $2^{\text{poly}(\log N, 1/\varepsilon, 1/\delta, k)}$ circuits A of that size, and write the corresponding function (or string of length N) A^G. In this list of strings, at least one of them has agreement at least $1 - \delta$ with $C(M)$. We can apply to each of these strings the decoding algorithm of C, and then we get another list of $2^{\text{poly}(\log N, 1/\varepsilon, 1/\delta, k)}$ strings, and one of them must be M.

We have thus described a list-decoding algorithm for C' that can correct a $1 - \varepsilon$ fraction of errors.

Unfortunately, the encoding length of C' is N^k, where N is the length of the encoding of C, and the list size is only quasi-polynomial in N.

1.2. Reducing the Encoding Length and List Size

In the above use of the Direct Product Lemma, the length of the new code is equal to the number of possible inputs for the function F, and the size of the list is the number of possible circuits A.

If we think of the circuit A as a fixed uniform machine taking advice (where the advice may depend both on f and G), then we only need to enumerate all possible advice strings, and only the number of advice bits matters to determine the list size. In fact, A may even be a probabilistic machine with advice, in a model where the advice string may depend on the random choices of A (but not on the input of A), as in [19]. Then we can pick randomness for A and then enumerate all possible advice strings (this approach leads to a probabilistic list-decoding algorithm).

In such a model, the proofs by Levin [12] and by Goldreich et al. [6] can be modified to yield a probabilistic algorithm A that needs only $\text{poly}(1/\varepsilon, 1/\delta)$ bits of advice, which gives a list-decoding algorithm with a list size that is independent of N. The encoding length, however, is still N^k.

In order to reduce the encoding length, we need to consider derandomized versions of the Direct Product Lemma. In the derandomized setting, the inputs $(x_1, \ldots, x_k)$ are chosen according to a pseudorandom distribution and an input for $F()$ or $G()$ is the seed used to generate $(x_1, \ldots, x_k)$ rather than the points themselves.

In the derandomized proof by Impagliazzo [10], the x_i are pairwise independent, so that the number of inputs of F is only $O(N^2)$. Unfortunately the proof is inherently non-uniform, and, in the coding application, the list size is again super-polynomial.

In the derandomized proof of Impagliazzo and Wigderson, there is a trade-off between the input length of F and the amount of non-uniformity. When F has $\text{poly}(N)$ inputs, the amount of non-uniformity is $N^{\Omega(1)}$, and it gets smaller for larger inputs.[1]

1 The main goal in [11] is to achieve $\varepsilon = 2^{-\Omega(\delta k)}$ while having the number of inputs for F polynomial in N for constant δ. In the application to pseudorandomness for which the result was meant, it was not a problem that the circuit A would have size $N^{\Omega(1)}$. The proof by Impagliazzo is "more derandomized" and uses less advice, but only achieves $\varepsilon \approx 1/(\delta k)$.

1.3. An Advice-efficient Proof of the Pairwise Independent Direct Product Lemma

The main result of this paper is a proof of the Direct Product Lemma for pairwise independent inputs in which the algorithm A in the reduction uses randomness and $\mathrm{poly}(k, 1/\varepsilon, 1/\delta)$ bits of advice.

The proof by Impagliazzo for the pairwise independent case has two steps. He first shows that for every function that is "weakly hard" on average there is a "hard core" subset of inputs on which the function is very hard on average. Then, he uses this result to prove the Direct Product lemma for pairwise independent inputs. We follow the same approach and we show show how to reduce advice in both parts of the proof.

The coding-theoretic application of our result is as follows. Starting from a code with binary codewords of length N and with a unique-decoding algorithm that corrects up to δN errors, we get a code with codewords of length N^2, over the alphabet $\{0,1\}^k$ and with a list-decoding algorithm that produces a list of size $2^{\mathrm{poly}(1/\varepsilon,1/\delta,1/k)}$ and corrects up to a fraction $1-\varepsilon$ of errors, where $\varepsilon = \Omega(1/(\delta k))$.

Starting from asymtpotically good codes, such as those of [15], where N is linear in the length of the message and δ is a constant, one gets codes with quadratic encoding length and quasi-linear list-decoding algorithm that can correct a $1-\varepsilon$ fraction of errors, producing a list of size $2^{\mathrm{poly}(1/\varepsilon)}$.

1.4. Comparison with [2, 7, 8, 9]

The codes obtained from Direct Product Lemmas (derandomized or not) can be seen as a special case of a general method to obtain error-correcting codes with large minimum distance from error-correcting codes with smaller minimum distance. The method was introduced by Alon et al. [2] and used recently by Guruswami and Indyk [7, 8, 9].

Suppose we have an error-correcting code $C : \mathcal{M} \to \{0,1\}^N$ in which we can correct a δ fraction of errors. Let $G = ([N], [N'], E)$ be a bipartite right-regular graph with N vertices on the left and N' vertices on the right, and let k be the degree of the vertices on the right. For a vertex v on the right and an index i, denote by $\Gamma_i(v)$ the i-th neighbor of v.

Then define a new code $C' : \mathcal{M} \to (\{0,1\}^k)^{N'}$ as follows.

$$C'(M)[v] = (C(M)[\Gamma_1(v)], \cdots, C(M)[\Gamma_k(v)])$$

The reader can verify that our construction for the case of the standard direct product corresponds to using the graph $G = ([N], [N]^k, E)$ where a vertex $(u_1, \ldots, u_k)$ on the right is adjacent to the vertices $u_1, \ldots, u_k$ on the left.

In the pairwise independent case the graph is $G = ([N], [N]^2, E)$ where a vertex (a, b) on the right is adjacent to the vertices $(a+b), (a+2b), \ldots, (a+kb)$ on the left.

We elaborate on this perspective in Section 4 and we show that a simple modification of one of the algorithms of Guruswami and Indyk leads to the completely derandomized and completely uniform direct product theorem.

1.5. Uniform Amplification of Hardness in NP

O'Donnell uses Impagliazzo's result about hard-core sets to prove a result about amplification of average-case complexity in NP. O'Donnell's result is as follows: suppose that for every problem L in NP there is a family of polynomial size circuits that solves L on a $1/2 + 1/n^{.5-\varepsilon}$ fraction of inputs; then for every balanced[2] problem L in NP there is a family of polynomial size circuits that solves L on a $1 - 1/\mathrm{poly}(n)$ fraction of inputs. If the assumption is strengthened to the existence, for every NP problem, of polynomial size circuits that solve L on a $1/2 + 1/n^{1/3-\varepsilon}$ fraction of the inputs, then the conclusion that for every NP problem L (not just for balanced ones) there is a family of circuits that solves L on a $1 - 1/\mathrm{poly}(n)$ fraction of inputs.

The result refers to circuits because the use of Impagliazzo's hard core sets makes the reduction non-uniform.

Using our version of Impagliazzo's result, we can prove a statement of the following form:

> Suppose that for every problem L in NP there is a probabilistic polynomial time algorithm that solves L on a $1/2+\varepsilon$ fraction of inputs[3] for every input length;
>
> Then for every problem L in NP there is a probabilistic algorithm that, given n, runs in $\mathrm{poly}(n) \cdot 2^{\mathrm{poly}(1/\varepsilon,1/\delta)}$ time, produces a list of $2^{\mathrm{poly}(1/\varepsilon,1/\delta)}$ circuits, and, with high probability, one of the circuits solves L on a $1-\delta$ fraction of inputs of length n.

When ε and δ are at least $1/(\log n)^c$ for a sufficiently small $c > 0$, then the list has polynomial size. We are then faced with the following task: given a list of circuits for an NP problem, such that one of the circuits works well on average, construct a single circuit that works well on average.

If the circuits solved the *search* version of L, then the task would be easy: run the circuits in parallel, then if at least one of them finds a certificate accept, otherwise reject.

A sensible idea would then be to use the search-to-decision reduction for average-case NP problem of Ben-

2 By balanced we mean that on every input length half of the instances are YES instances and half of the instances are NO instances.

3 By this, we mean that there is an algorithm A such that for a random x and a random choice of randomness for A there is a probability $1/2+\varepsilon$ that A is correct on x.

David and others [3]. The procedure starts from a language L and defines a new language L' such that a good-on-average *decision* algorithm for L' yields a good-on-average *search* algorithm for L. The problem is that the reduction transforms an algorithm for L' that works on a $1-\delta$ fraction of inputs of length $n'(n)$ into an algorithm for L that works on a $1 - O(\delta \cdot m(n))$ fraction of inputs of length n, where $m(n)$ is the length of witnesses for instances of L of length n and $n'(n)$ is polynomial in n. In particular, the reduction gives nothing if applied to an algorithm for L' that succeeds only on a $1 - 1/\mathrm{poly}\log n$ fraction of inputs.

We do not know how to overcome this difficulty, but we show a way of removing non-uniformity under the stronger assumption that for every NP problem we have an an algorithm that solves it on slightly more than a $3/4$ fraction of inputs.

Our final result is that if for every problem L in NP there is a probabilistic polynomial time algorithm that solves L on a $3/4 + (1/\log n)^c$ fraction of inputs of length n; then for every problem L in NP there is a probabilistic polynomial time algorithm that solves L on a $1 - 1/(\log n)^c$ fraction of inputs, where c is an absolute constant.

1.6. Overview

We present an almost uniform version of Impagliazzo's hard-core set result in Section 2. We use this result in Section 3 to prove an advice-efficient direct product lemma for pairwise indpendent inputs. In Section 4 we point out the equivalence between certain graph-based constructions of error-correcting codes and the constructions derived from derandomized direct product lemmas. Finally, in Sections 5 and 6 we give a uniform amplification of hardness result for NP.

Some proofs are omitted due to space constraints. A full version of this paper, which includes all proofs, is available from ECCC [18].

2. An Advice-efficient Version of Impagliazzo's Hard-Core Sets

We can abstract Impagliazzo's main result in his construction of hard-core sets as follows.

Definition 1 (Set-Function Game). *Let N be an integer, $\delta, \varepsilon \in (0, 1/2)$ be fractions and $f : [N] \to \{0,1\}$ be a function. Consider the following game: at every step i the first player produces a set $H_i \subseteq [N]$ such that $|H_i| \geq \delta N$ and the second player replies with a function $g_i : [N] \to \{0,1\}$ such that g_i and f agree on at least an $1/2+\varepsilon$ fraction of elements of H_i. The first player wins at step i if the function $g(x) = \mathrm{majority}\{g_1(x), \ldots, g_i(x)\}$ agrees with $f()$ on at least a $1-\delta$ fraction of the elements of $[N]$.*

Lemma 2 (Impagliazzo [10]). *There is a strategy for the first player such that for every strategy for the second player involving $2^{N^{o(1)}}$ possible functions g, the first player wins within $\mathrm{poly}(1/\varepsilon, 1/\delta)$ steps.*

Impagliazzo uses Lemma 2 in the following way. Suppose that we have a function $f : \{0,1\}^n \to \{0,1\}$ such that for every set $H \subseteq \{0,1\}^n$ with $|H| \geq \delta 2^n$ there is a circuit C of size $\leq s$ such that C computes f on at least a $1/2 + \varepsilon$ fraction of the elements of H. Then, we can play the set-function game (as a mental experiment) with the first player using the strategy of Lemma 2 and the second player always replying with a circuit of size $\leq s$. Then, we conclude that f can be computed on a $1-\delta$ fraction of the inputs by a circuit of size $s \cdot \mathrm{poly}(1/\varepsilon, 1/\delta)$. By contrapositive, if it is impossible to compute f on a $1-\delta$ fraction of inputs using a circuit of size s, then there must be a set $H \subseteq \{0,1\}^n$ such that no circuit of size $s \cdot \mathrm{poly}(\varepsilon, \delta)$ can compute f on more than a $1/2 + \varepsilon$ fraction of the elements of H.[4]

The argument is inherently non-uniform, and, in fact, it is not clear how to use Lemma 2 in a uniform setting. The following Lemma gives a way of doing it.

Lemma 3. *Let $\mathcal{C}$ be a distribution of circuits samplable in time t, $f : [N] \to \{0,1\}$ be a function and let $\gamma, \delta, \varepsilon$ be such that for every subset $H \subseteq [N]$, $|H| \geq \delta N$, we have*

$$\Pr_{C \leftarrow \mathcal{C}} \left[\Pr_{x \in H} [f(x) = C(x)] \geq \frac{1}{2} + \varepsilon \right] \geq \gamma$$

Then there is a distribution $\mathcal{C}'$ of circuits samplable in time $t \cdot \mathrm{poly}(1/\varepsilon, 1/\delta)$ such that

$$\Pr_{C \leftarrow \mathcal{C}'} \left[\Pr_{x \in [N]} [f(x) = C(x)] \geq 1 - \delta \right] \geq \gamma^{\mathrm{poly}(1/\varepsilon, 1/\delta)}$$

Proof. As a mental experiment, we are going to run Impagliazzo's procedure against samples from $\mathcal{C}$. We need at most $t = \mathrm{poly}(1/\varepsilon, 1/\delta)$ circuits, and, each time, we have a probability γ that a circuit sampled from $\mathcal{C}$ is a legal move.

The distribution $\mathcal{C}'$ is thus as follows. We compute the upper bound $t = \mathrm{poly}(1/\varepsilon, 1/\delta)$ to the number of moves in the strategy of Lemma 2. Then we pick at random $i \in \{1, \ldots, t\}$ and sample i circuits $C_1, \ldots, C_i$ independently from $\mathcal{C}$. Finally, we output the circuit $C(x) = \mathrm{majority}\,\{C_1(x), \ldots, C_i(x)\}$.

There is a probability at least $(1/t) \cdot \gamma^t$ that C computes f on a $1-\delta$ fraction of inputs. □

The conclusion of the Lemma can be equivalently stated as the existence of a probabilistic algorithm that produces

4 The Set-Function game in [10] refers not to sets of size δN, but to "measures," or, essentially, distributions of min-entropy at least $\log(\delta N)$. However Impagliazzo also shows a result about "rounding" measures to sets which implies that Lemma 2 is true as stated.

a list of $(1/\gamma)^{\mathrm{poly}(1/\varepsilon,1/\delta)}$ circuits such that with high probability one of them solves f on a $1-\delta$ fraction of inputs. The correct circuit can then be specified using $\log(1/\gamma)\cdot\mathrm{poly}(1/\varepsilon,1/\delta)$ bits of advice, assuming a non-uniform model like the one of [19], where a randomized machine tosses its random coins, and then receives an advice that depends on the randomness and on the input length, but not on the input itself.

3. Advice-efficient Direct Product Lemma for Pair-wise Independent Inputs

For simplicity, in this section we will refer to a specific pair-wise independent generator, even though the argument could be applied more generally.

Suppose we have a function $f:[N]\to\{0,1\}$ and that $[N]$ is a field (for example, N is prime and we do operations $\pmod N$). Then, if we pick $a,b\in[N]$ at random, the elements $a+b, a+2b, \ldots, a+kb$ are pairwise independent.

Define the function $f^k:[N]^2\to\{0,1\}^k$ as

$$f^k(a,b)=f(a+b),f(a+2b),\ldots,f(a+kb)\ .$$

Then the pair-wise independent Direct Product Lemma of Impagliazzo [10] implies that if f is hard to compute on more than a $1-\delta$ fraction of inputs with circuits of size s, then f^k is hard to compute on more than an ε fraction of inputs with circuits of size $s\cdot\mathrm{poly}(1/k,\delta)$, where $\varepsilon=O(1/\delta k)$.

The proof works as follows: suppose that there is a circuit A of size s that computes f^k on more than an ε fraction of inputs; then for every $H\subseteq[N]$ we show that there is a circuit that computes f on a $1/2+\Omega(\varepsilon)$ fraction of the elements of H, furthermore, C is of size $s+\mathrm{poly}\log N$. Using the results about hard-core sets, we conclude that there is a circuit of size $s\cdot\mathrm{poly}(k,1/\delta)$ that computes f on a $1-\delta$ fraction of inputs.

Let us now see the proof in detail. The following presentation is taken from [10] by specializing the analysis given for general r-wise independent generators to the case of the particular pairwise independent generator considered in this section. At every step we show how to replace non-uniform choices with random choices, and estimate the probability that random choices are correct.

Assume $\varepsilon\geq 128/\delta k$ and let $\gamma=\varepsilon/256$.[5] Let $A:[N]^2\to\{0,1\}^k$ be a function that agrees with f^k on at least an ε fraction of the inputs. Let $A_i(a,b)$ be the i-th output of A. Fix a set $H\subseteq[N]$ such that $|H|=\delta N$. Our goal is to find a circuit C such that C computes f on at least a $1/2+\gamma$ fraction of the elements of $[N]$ and such that the size of C is not much larger than the size of A.

5 We are making no attempt to optimize constants.

We will give a probabilistic procedure that succeeds with probability $\mathrm{poly}(1/\varepsilon,1/\delta,1/k)$.

Define the k random variables $F_1,\ldots,F_k$ as follows: pick at random $a,b\in[N]$ then

- $F_i=1$ if $a+ib\in H$ and $A_i(a,b)=f(a+ib)$,
- $F_i=0$ otherwise.

A first observation is that if there is an i such that $\mathbf{Pr}[F_i=1]\notin\delta(1/2\pm\gamma)$ then our task of constructing C is quite easy.

Claim 1. *Suppose that $\mathbf{Pr}[F_i=1]>\delta(1/2+\gamma)$ or $\mathbf{Pr}[F_i=1]<\delta(1/2+\gamma)$. Then there is a circuit of size $\mathrm{poly}\log N$ that makes one oracle query to A and solves f on a $1/2+\gamma$ fraction of H. Furthermore, there is a probabilistic construction of circuits of size $\mathrm{poly}\log N$ that make one oracle query to A. With probability at least $\gamma/2$, the construction gives a circuit solves f on a $1/2+\gamma/2$ fraction of H.*

It remains to consider the case in which $\delta(1/2-\gamma)\leq\mathbf{E}[F_i]\leq\delta(1/2+\gamma)$ for every i. We note that, of course, $\mathbf{E}[\sum_i F_i]=\delta k(1/2\pm\gamma)$. On the other hand, we are going to argue that $\sum_i F_i$ is not very concentrated around its expectation, and so the F_i cannot be almost pairwise independent.

Claim 2. *Suppose $\mathbf{Pr}[F_i=1]=\delta(1/2\pm\gamma)$ for all i. Then there are indices i,j such that*

$$\mathbf{E}[F_iF_j]-(\mathbf{E}[F_i]\,\mathbf{E}[F_j])\geq\frac{\delta^2\varepsilon}{64}$$

The dependency can be exploited to compute f on H.

Claim 3. *Under the same assumption of Claim 2, there is a circuit of $\mathrm{poly}\log N$ size that makes one oracle query to A and solves f on a $1/2+4\gamma/3$ fraction of H. There is also a samplable distribution of circuits of size $\mathrm{poly}\log N$ with one oracle query to A such that the distribution produces with probability at least $\gamma/6$ a circuit that solves f on a $1/2+\gamma$ fraction of H.*

See [18] for the proof of Claims 1, 2 and 3.

Finally, consider the following distribution over circuits: pick at random indices i,j, then with probability 1/2 sample a circuit from the distribution of Claim 1 (with respect to i) and with probability 1/2 sample a circuit from the distribution of Claim 3 (with respect to i,j).

Then, in each possible case, there is a probability $\Omega(\delta\varepsilon/k^2)$ that we sample a circuit computes f on at least a $1/2+\gamma/2$ fraction of the elements of H. Notice that the distribution is independent of H, and, indeed, the result is true for all H such that $|H|=\delta N$.

We can then apply Lemma 3. We summarize the result proved in this section as follows.

Theorem 4. *Let $f : [N] \to \{0,1\}$, and let ε, δ, k, and $f^k : [N]^2 \to \{0,1\}^k$ be defined as the beginning of this section. Let A be a function that has agreement ε with f^k. There is a probabilistic algorithm that runs in $2^{\text{poly}(1/\varepsilon,1/\delta)} \cdot \text{poly}\log N$ time and produces a list of $2^{\text{poly}(1/\varepsilon,1/\delta)}$ oracle circuits; each circuit has size $\text{poly}(\log N, 1/\varepsilon, 1/\delta)$ and makes $\text{poly}(1/\varepsilon, 1/\delta)$ oracle queries. With high probability, at least one circuit in the list, when given oracle access to A, solves f on a $1-\delta$ fraction of inputs.*

For the coding-theoretic application, let $C : \mathcal{M} \to \{0,1\}^N$ be an error-correcting code with a linear-time decoding algorithm that can correct a δ fraction of errors and with a quadratic time (or better) encoding algorithm. For example, C could be a Sipser-Spielman code [15]. Assume N is prime, and consider $[N]$ as a field.

Then, for parameters ε, k as the beginning of this section, define the code $C' : \mathcal{M} \to (\{0,1\}^k)^{N^2}$. For a message M and for indices $a, b \in [N]$, the entry indexed by (a,b) in $C'(M)$ is the k-bit string

$$C'(M)[a,b] = (C(M)[a+b] \cdots C(M)[a+kb])$$

Let A be a string that has agreement ε with $C'(M)$. Then there is a probabilistic algorithm that runs in $2^{\text{poly}(1/\varepsilon,1/\delta)} \cdot \text{poly}\log N$ time and produces a list of $2^{\text{poly}(1/\varepsilon,1/\delta)}$ oracle circuits; each circuit has size $\text{poly}(\log N, 1/\varepsilon, 1/\delta)$ and makes $\text{poly}(1/\varepsilon, 1/\delta)$ oracle queries. With high probability, at least one circuit in the list, when given oracle access to A, defines a string that agrees with $C(M)$ on a $1-\delta$ fraction of entries.

In $2^{\text{poly}(1/\varepsilon,1/\delta)} \cdot N\text{poly}\log N$ time we can compute all the strings defined by all the circuits in the list, and then apply the unique decoding algorithm to each of them. This way we get a list of size $2^{\text{poly}(1/\varepsilon,1/\delta)}$ that contains M.

4. Codes and Direct Product Lemmas From Expander Graphs

As discussed in the introduction, codes obtained from a Direct Product Lemma can be seen as graph-based constructions of codes in the spirit of [2, 7, 8, 9]. In fact the converse is also true: if a graph-based construction of codes has a sub-linear time error-correction procedure, then it also gives a direct product result. In this section we observe that the decoding procedure for one of the codes in [8] has indeed a sub-linear time decoding procedure, and hence we derive a new direct product result from it. The result is weak in the sense that it proves only constant average-case hardness, but the result is *completely* derandomized, and the reduction is uniform and deterministic.

Consider the following definition.

Definition 5. *We say that a k-regular bipartite graph $G = ([N],[N],E)$ is an (ε,δ)-mixer if for every subset B of vertices on the right such that $|B| \geq (1/2-\varepsilon)N$, there are at most δN vertices v on the left such that $|\Gamma(v) \cap B| > k/2$.*

Lemma 6. *There are explicit (ε,δ)-mixers with $k = \text{poly}(1/\varepsilon, 1/\delta)$.*

In fact, any family of expanders with a $\text{poly}(1/\varepsilon, 1/\delta)$ eigenvalue gap is an (ε,δ) mixer. In the Lemma above, "explicit" means that the neighborhood of a vertex can be computed in time polynomial in $\log N$ and in k. The constructions in [13], and, in fact, even those in [5], prove Lemma 6.

Theorem 7. *Let $f_n : \{0,1\}^n \to \{0,1\}$, $\varepsilon(n), \delta(n) > 0$ be arbitrary, and $G_n = (\{0,1\}^n, \{0,1\}^n, E)$ be a an explicit family of $(\varepsilon(n), \delta(n))$ mixers of degree $k(n)$.*

Define the function $F_n : \{0,1\}^n \to \{0,1\}^{k(n)}$ as

$$F(x) = (f(\Gamma_1(x)), \cdots, f(\Gamma_k(x))$$

Suppose there is a deterministic algorithm A running in time $t(n)$ that solves F_n on a $1/2+\varepsilon(n)$ fraction of the inputs.

Then there is a deterministic algorithm A' running in time $O(t \cdot poly(k,n))$ that solves f_n on a $1-\delta(n)$ fraction of inputs.

Notice that the new function has the *same input length* as the original function, and the result preserves both uniformity and determinism. This is probably the only known "amplification of hardness" proved with a deterministic reduction.

Proof. On input $x \in \{0,1\}^n$, algorithm $A'()$ finds the k neighbors $y_1, \ldots, y_k$ of x in G_n and applies algorithm $A()$ to each of them. The results $A(y_1), \ldots, A(y_k)$ contain each a prediction of the value $f(x)$. Algorithm A' returns the value that occurs more often among such predictions.

Let $B \subseteq \{0,1\}^n$ be the set of inputs y such that $A(y) \neq F(y)$. By assumption, $|B| \leq (1/2-\varepsilon)2^n$. The inputs $x \in \{0,1\}^n$ such that $A'(x) \neq f(x)$ are such that a majority of their neighbors belong to B. By the mixing property of G, there can be at most $\delta 2^n$ such inputs x. $\square$

By applying a randomness-efficient version of Goldreich-Levin, we can also define a boolean function $f' : \{0,1\}^{n+\log k + O(\log 1/\varepsilon)} \to \{0,1\}$ such that a deterministic algorithm that computes f' on a $7/8 + O(\varepsilon)$ fraction of inputs can be turned into a deterministic algorithm of comparable complexity that computes f on a $1-\delta$ fraction of inputs, where $n, k, \varepsilon, \delta$ are as in the above Theorem.

5. Advice-Efficient Amplification of Hardness in NP

In this section we present the results of [14] and explain how to use Lemma 3 instead of Impagliazzo's hard core sets in order to derive a uniform reduction.

5.1. O'Donnell's Proof

Let $f : \{0,1\}^n \to \{0,1\}$ be an NP function and $g : \{0,1\}^k \to \{0,1\}$ be a monotone function. Then $f'(x_1,\ldots,x_k) = g(f(x_1),\ldots,f(x_k))$ is still an NP function. O'Donnell shows that for a proper choice of g and for a certain range of parameters ε and δ, if there is a circuit A of size s that solves f' on an $1/2+\varepsilon$ fraction of inputs then, for every set H of density δ, there is a circuit C of size $s+\mathrm{poly}(k,n)$ that solves f on a $1/2+\Omega(\varepsilon/\sqrt{k})$ fraction of inputs from H. In turn, this implies that there is a circuit of size $s\cdot\mathrm{poly}(1/\varepsilon,1/\delta,k)$ that solves f on a $1-\delta$ fraction of inputs.

We sketch below a slightly different, and simpler, argument that shows, under the same assumption, that for every set H of density δ there is a circuit of size $s+\mathrm{poly}(k,n)$ that solves f on a $1/2+\varepsilon/(\delta k)$ fraction of inputs H. Our argument is easier to adapt to the uniform setting.[6]

Let $f : \{0,1\}^n \to \{0,1\}$ be a function (that we think of as being hard to compute on more than a $1-\delta$ fraction of inputs) and let $H \subseteq \{0,1\}^n$ be a set of size $\delta 2^n$ (that we think of as being a hard-core set for f).

Let us define a random function $b : \{0,1\}^n \to \{0,1\}$ as follows: for $x \notin H$, $b(x) = f(x)$, while, for $x \in H$, $b(x)$ outputs a random bit. If H is a hard-core set for f, then the distributions $(x, f(x))$ and $(x, b(x))$ are computationally indistinguishable. More precisely, if we are given a circuit of size s that has distinguishing probability ε between the two distributions, then we get a circuit of size $s+O(1)$ that computes f on at least a $1/2+\varepsilon/\delta$ fraction of the elements of H.

An hybrid argument shows that the distributions

$$x_1, x_2, \ldots, x_k, f(x_1), f(x_2), \ldots, f(x_k)$$

and

$$x_1, x_2, \ldots, x_k, b_1(x_1), b_2(x_2), \ldots, b_k(x_k)$$

are also computationally indistinguishable, if H is a hard-core set. (The functions b_i are independent copies of the function b that we previously defined.) More precisely, if we are given a circuit of size s that has distinguishing probability ε between the two distributions, then we get a circuit of size $s+O(nk)$ that computes f on at least a $1/2+\varepsilon/(\delta k)$ fraction of the elements of H.

6 A similar argument appears in [17].

If $g : \{0,1\}^k \to \{0,1\}$ is a function computable by a circuit of size $\mathrm{poly}(k)$, then the distributions

$$x_1, x_2, \ldots, x_k, g(f(x_1), f(x_2), \ldots, f(x_k))$$

and

$$x_1, x_2, \ldots, x_k, g(b_1(x_1), b_2(x_2), \ldots, b_k(x_k))$$

are still computationally indistinguishable if H is a hard-core set.

O'Donnell shows that if f is a balanced function, then for every ε, δ there is a $k = \mathrm{poly}(1/\varepsilon, 1/\delta)$ and a function $g : \{0,1\}^k \to \{0,1\}$ computable by a circuit of size $O(k)$ such that the distributions

$$x_1, x_2, \ldots, x_k, g(b(x_1), b(x_2), \ldots, b(x_k))$$

and

$$x_1, x_2, \ldots, x_k, r$$

(where r is a random bit) have statistical distance at most ε. Notice that this is a purely information-theoretic result, that uses only the fact that f is balanced and that H has density δ.

Then we get that the distributions

$$x_1, x_2, \ldots, x_k, g(f(x_1), f(x_2), \ldots, f(x_k))$$

and

$$x_1, x_2, \ldots, x_k, r$$

are computationally indistinguishable, which means that the function $f'(x_1,\ldots,x_k) = g(f(x_1), f(x_2), \ldots, f(x_k))$ is very hard on average.

5.2. Our Result

In this section we prove a version of O'Donnell result with bounded non-uniformity.

We will proceed as in [14], with a careful use of non-uniformity. First, we state formally the information-theoretic part of the argument of [14].

Theorem 8 ([14]). *For every $\varepsilon, \delta > 0$ there is a $k = \mathrm{poly}(1/\varepsilon, q/\delta)$ and a function $g : \{0,1\}^k \to \{0,1\}$ such that the following holds. Let $f : \{0,1\}^n \to \{0,1\}$ be a function ε/k-close to balanced, $H \subseteq \{0,1\}^n$ be of density δ, $b_1^f, \ldots, b_k^f : \{0,1\}^n \to \{0,1\}$ be independent random functions such that $b_i^f(x) = f(x)$ for $x \notin H$ and $b_i^f(x)$ outputs a random bit if $x \in H$. Then the distributions*

$$x_1, \ldots, x_k, g(b_1^f(x_1), \ldots, b_k^f(x))$$

and

$$x_1, \ldots, x_k, r$$

have statistical distance at most 2ε, where the x_i are uniform and independent in $\{0,1\}^n$ and r is uniform in $\{0,1\}$.

The proof of Theorem 8 uses the following notion.

Definition 9. *For a string $x \in \{0,1\}^k$, denote by $N_\delta(x)$ the random variable obtained by flipping each bit of x independently with probability δ.*

The noise stability *of a function $g : \{0,1\}^k \to \{0,1\}$ at parameter δ, denoted $\textsc{NoiseStab}_\delta(g)$ is defined as*

$$\textsc{NoiseStab}_\delta(g) = \mathbf{Pr}[g(x) = g(N_\delta(x))]$$

where x is uniform.

The following two results give a proof of Theorem 8.

Lemma 10 ([14]). *For every $\varepsilon, \delta > 0$ there is a $k = \mathrm{poly}(1/\varepsilon, 1/\delta)$ and a function $g : \{0,1\}^k \to \{0,1\}$ such that $\textsc{NoiseStab}_{\delta'}(g) \leq 1/2 + \varepsilon$ for every $\delta \leq \delta' < 1$.*

Lemma 11 ([14]). *Let $f : \{0,1\}^n \to \{0,1\}$ be a balanced function, $H \subseteq \{0,1\}^n$ be of density δ, $b_1^f, \ldots, b_k^f : \{0,1\}^n \to \{0,1\}$ be independent random functions such that $b_i^f(x) = f(x)$ for $x \notin H$ and $b_i^f(x)$ outputs a random bit if $x \in H$. Let $g : \{0,1\}^k \to \{0,1\}$ be a function such that $\textsc{NoiseStab}_{\delta/2}(g) \leq 1/2 + 2\varepsilon^2$. Then the distributions*

$$x_1, \ldots, x_k, g(b_1^f(x_1), \ldots, b_k^f(x))$$

and

$$x_1, \ldots, x_k, r$$

have statistical distance at most ε, where the x_i are uniform and independent in $\{0,1\}^n$ and r is uniform in $\{0,1\}$.

In order to prove Theorem 8, we just need to take into account the fact that f is not perfectly balanced. See [18] for more details.

Now we are left with the reduction to the information-theoretic case.

Lemma 12. *Let $g : \{0,1\}^k \to \{0,1\}$ be as in Theorem 8, $f : \{0,1\}^n \to \{0,1\}$ be ε/k-close to balanced and A be a polynomial-time algorithm such that*

$$\mathbf{Pr}[A(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k))] \geq \frac{1}{2} + 3\varepsilon$$

Then there is a polynomial time samplable distribution of circuits $\mathcal{C}$ such that for every set $H \subseteq \{0,1\}^n$ of density δ we have

$$\mathop{\mathbf{Pr}}_{C \sim \mathcal{C}}\left[\mathop{\mathbf{Pr}}_{x \in H}[C(x) = f(x)] \geq \frac{\varepsilon}{2\delta k}\right] \geq \frac{\varepsilon}{\delta k^2 \cdot 2^{2k-1}}$$

The proof proceed as in the overview of Section 5.1. See [18] for a complete proof.

We get our amplification of hardness result by using the above lemma and Lemma 3 (our uniform version of Impagliazzo's hard core result).

Lemma 13. *Let $g : \{0,1\}^k \to \{0,1\}$ be as in Theorem 8, $f : \{0,1\}^n \to \{0,1\}$ be ε/k-close to balanced and A be a polynomial-time algorithm such that*

$$\mathbf{Pr}[A(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k))] \geq \frac{1}{2} + 2\varepsilon$$

Then there is a polynomial time samplable distribution of circuits $\mathcal{C}$ such that

$$\mathop{\mathbf{Pr}}_{C \sim \mathcal{C}}\left[\mathop{\mathbf{Pr}}_{x \in H}[C(x) = f(x)] \geq 1 - \delta\right] \geq 2^{-\mathrm{poly}(1/\varepsilon, 1/\delta)}$$

Finally, we want to generalize the result from the almost-balanced case to the general case. We can do so by using the same padding technique adopted in [14]. The reader is referred to [14] for details.

Theorem 14. *For every polynomial time computable functions ε, δ the following holds.*

Suppose that for every language L in NP there is a probabilistic algorithm A such that for every input length n we have

$$\mathop{\mathbf{Pr}}_{x \in \{0,1\}^n}[A(x) = L(x)] \geq \frac{1}{2} + \varepsilon(n)$$

Then for every language L in NP there is a probabilistic algorithm A' that on input n runs in $\mathrm{poly}(n, 1/\varepsilon(n), 1/\delta(n))$ time and outputs a circuit, and

$$\mathop{\mathbf{Pr}}_{C \sim A'(n)}\left[\mathop{\mathbf{Pr}}_{x \in \{0,1\}^n}[C(x) = L(x)] \geq 1 - \delta(n)\right] \geq 2^{-\mathrm{poly}(1/\varepsilon(n), 1/\delta(n))}$$

Notice that the conclusion of the theorem could be stated equivalently as the existence of a probabilistic algorithm running in time polynomial in n and in $2^{\mathrm{poly}(1/\varepsilon, 1/\delta)}$ that produces a list of $2^{\mathrm{poly}(1/\varepsilon, 1/\delta)}$ circuits such that one of them solves f on at least a $1 - \delta$ fraction of inputs.

6. Uniform Amplification of Hardness in NP

In this section we prove the following result.

Theorem 15. *There is a constant $c > 0$ such that the following is true. Suppose that for every NP problem there is a probabilistic polynomial time algorithm that, on every input length, succeeds on a $3/4 + 1/(\log n)^c$ fraction of inputs. Then for every balanced NP problem there is a probabilistic polynomial time algorithm that, on every input length, succeeds on a $1 - 1/(\log n)^c$ fraction of inputs.*

Given the results of the previous section, the assumption of Theorem 15 implies that, for every language L in NP, we have a probabilistic algorithm that, on input n, runs in $\mathrm{poly}(n)$ time and produces a list of $\mathrm{poly}(n)$ circuits of $\mathrm{poly}(n)$ size such that at least one of them solves L on at least a $1-\delta$ fraction of inputs, where $\delta = 1/(\log n)^{\Omega(1)}$. We would like to be able to recognize such a circuit. Unfortunately, we do not know how to do that. On the other hand, under the assumption of the theorem, we are able to solve a slightly weaker promise problem, and solving the promise problem will be enough to prove the conclusion of the theorem.

Theorem 16. *For every function $\delta(n)$ there are function $\gamma(n), \sigma(n)$ such that $\gamma(n) \leq \delta(n)^{\Omega(1)}$ and $\sigma(\mathrm{poly}(n)) \leq \delta(n)^{\Omega(1)}$ and the following happens.*

Suppose that for every balanced language L in NP there is a probabilistic polynomial time algorithm that solves L on a $3/4 + \sigma(n)$ fraction of the inputs of length n.

Then for every balanced language L in NP there is a probabilistic polynomial time probabilistic algorithm A such that

- *For every circuit C that solves L on at least a $1 - \delta(n)$ fraction of the inputs of length n, $\mathbf{Pr}[A(C)$ accepts $] \geq 3/4$.*
- *For every circuit C that solves L on fewer than a fraction $1 - \gamma(n)$ of the inputs of length n, $\mathbf{Pr}[A(C)$ accepts $] \leq 1/4$.*

Of course the probability of success can be amplified from $3/4$ to $1 - 2^{-n}$. See [18] for the proof. The proof of Theorem 15 is now mostly a matter of fixing parameters. We fix a c' small enough so that, in Theorem 14, under the assumption that every problem in NP can be solved on a $3/4$ fraction of inputs, then for every problem L in NP there is an algorithm A_{list} that constructs in polynomial time a list of circuits such that at least one of them solves the problem on a $1 - 1/(\log n)^{c'}$ fraction of inputs. Then we apply Theorem 16 with $\delta = 1/(\log n)^{c'}$, and we see that there are functions $\sigma, \gamma = \delta^{\Omega(1)}$ such that if every problem in NP can be solved uniformly on a $3/4 + \sigma$ fraction of inputs then for every problem L in NP there is an algorithm A_{dist} that is able to distinguish circuits that solve the problem on $\geq 1 - \delta$ fraction of inputs from circuits that solve the problem on $\leq 1 - \gamma$ fraction of inputs.

Finally, we get a good uniform algorithm for L as follows. On input x of length n, we run $A_{\mathrm{list}}(n)$ to get a list of possible circuits, then we run algorithm A_{dist} on each of the circuits. Let C be the first circuit that is accepted by A_{dist}; then we output $C(x)$. Since the generation of the circuit was independent of x, and since with high probability the circuit solves L on a $1 - \gamma$ fraction of inputs, then the whole algorithm has a success probability close to $1 - \gamma$. To complete the proof of Theorem 15 we just need to choose c small enough so that $\sigma(n), \gamma(n) \leq 1/(\log n)^c$ for sufficiently large n in the above argument.

7. Conclusions

Guruswami and Indyk [7, 8, 9] present, among several other results, error-correcting codes with linear-time encoding algorithms and linear time list-decoding algorithms that correct a $1 - \varepsilon$ fraction of errors. As in our case, their construction starts from codes with a unique decoding algorithm.

The Guruswami-Indyk construction in [9] is better than ours in any respect: shorter encoding length, faster encoding time, comparable decoding time and list size.

Sudan's [16] list-decoding algorithm for Reed-Solomon codes gives codes that decode a $1-\varepsilon$ fraction of errors while creating a list of size only $\mathrm{poly}(1/\varepsilon)$. Furthermore, using concatenation, one can have alphabets of size $\mathrm{poly}(1/\varepsilon)$ (while in our case the alphabet has size $2^{\mathrm{poly}(1/\varepsilon)}$) and quasi-linear or linear encoding length. The decoding time is polynomial in [16], but it has been improved to quasi-linear in [4, 1].

The main point of this paper, therefore, is not to present an improved, or even competitive, construction of list-decodable codes, but rather to show that results from complexity theory, with little manipulation, yield codes with reasonably non-trivial properties. Quite possibly, a new technique could be extracted by better understanding the algorithm presented in this paper, which could be used more directly to devise improved list-decoding algorithms. In any case, this connection gives a new way of looking at the list-decoding problem, and a new language to describe and analyze algorithms.

Within complexity theory, the observations and results of this paper are a challenge to come up with a direct product theorem for *almost pairwise independent inputs*. Such a result would give codes with quasi-linear, and possibly linear, encoding length.

The Direct Product Lemma should be provable with only $O(\log(1/\varepsilon\delta))$ bits of advice, which would lead to list-decoding algorithm with near-optimal list size. It is an interesting open question to prove such a result.

There is a lot of room for improvement in our uniform amplification of hardness result for NP. One approach to try and improve our results would be to devise an even more advice-efficient version of Impagliazzo's results.

Acknowledgements

I thank Madhu Sudan, Ryan O'Donnell, Salil Vadhan and Venkatesan Guruswami for helpful discussions.

References

[1] M. Alekhnovich. Linear diophantine equations over polynomials and soft decoding of reed-solomon codes. In *Proceed-*

ings of the 43rd IEEE Symposium on Foundations of Computer Science, pages 439–448, 2002.

[2] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. Construction of asymptotically good, low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 28:509–516, 1992.

[3] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992.

[4] G. Feng. Two fast algorithms in the sudan decoding procedure. In *Proceedings of the 37th Allerton Conference*, pages 545–554, 1999.

[5] O. Gabber and Z. Galil. Explicit construction of linear sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–425, 1981.

[6] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR lemma. Technical Report TR95-50, Electronic Colloquium on Computational Complexity, 1995.

[7] V. Guruswami and P. Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 658–667, 2001.

[8] V. Guruswami and P. Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smalle alphabets. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 812–821, 2002.

[9] V. Guruswami and P. Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003.

[10] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[11] R. Impagliazzo and A. Wigderson. $P = BPP$ unless E has sub-exponential circuits. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 220–229, 1997.

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

[13] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[14] R. O'Donnell. Hardness amplification within np. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 751–760, 2002.

[15] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. Preliminary version in *Proc. of FOCS'94*.

[16] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997. Preliminary version in *Proc. of FOCS'96*.

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

[18] L. Trevisan. List decoding using the XOR lemma. Technical Report TR03-042, Electronic Colloquium on Computational Complexity, 2003.

[19] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th IEEE Conference on Computational Complexity*, pages 129–138, 2002.