Approximately List-Decoding Direct Product Codes and Uniform Hardness Amplification

Russell Impagliazzo
University of California, San Diego
La Jolla, CA, USA
russell@cs.ucsd.edu

Ragesh Jaiswal
University of California, San Diego
La Jolla, CA, USA
rjaiswal@cs.ucsd.edu

Valentine Kabanets
Simon Fraser University
Vancouver, BC, Canada
kabanets@cs.sfu.edu

Abstract

We consider the problem of approximately locally list-decoding direct product *codes. For a parameter k, the k-wise direct product encoding of an N-bit message msg is an N^k-length string over the alphabet $\{0,1\}^k$ indexed by k-tuples $(i_1,\ldots,i_k)\in\{1,\ldots,N\}^k$ so that the symbol at position $(i_1,\ldots,i_k)$ of the codeword is $msg(i_1)\ldots msg(i_k)$. Such codes arise naturally in the context of hardness amplification of Boolean functions via the Direct Product Lemma (and the closely related Yao's XOR Lemma), where typically $k \ll N$ (e.g., $k = \mathsf{poly}\log N$).*

We describe an efficient randomized algorithm for approximate local list-decoding of direct product codes. Given access to a word which agrees with the k-wise direct product encoding of some message msg in at least an ϵ fraction of positions, our algorithm outputs a list of $\mathsf{poly}(1/\epsilon)$ Boolean circuits computing N-bit strings (viewed as truth tables of $\log N$-variable Boolean functions) such that at least one of them agrees with msg in at least $1-\delta$ fraction of positions, for $\delta = O(k^{-0.1})$, provided that $\epsilon = \Omega(\mathsf{poly}(1/k))$; the running time of the algorithm is polynomial in $\log N$ and $1/\epsilon$. When $\epsilon > e^{-k^{\alpha}}$ for a certain constant $\alpha > 0$, we get a randomized approximate list-decoding algorithm that runs in time quasi-polynomial in $1/\epsilon$ (i.e., $(1/\epsilon)^{\mathsf{poly}\log 1/\epsilon}$).

By concatenating the k-wise direct product codes with Hadamard codes, we obtain locally list-decodable codes over the binary alphabet, which can be efficiently approximately list-decoded from fewer than $1/2-\epsilon$ fraction of corruptions as long as $\epsilon = \Omega(\mathsf{poly}(1/k))$. As an immediate application, we get uniform *hardness amplification for $\mathsf{P}^{\mathsf{NP}_{\parallel}}$, the class of languages reducible to NP through one round of parallel oracle queries: If there is a language in $\mathsf{P}^{\mathsf{NP}_{\parallel}}$ that cannot be decided by any BPP algorithm on more that $1-1/n^{\Omega(1)}$ fraction of inputs, then there is another language in $\mathsf{P}^{\mathsf{NP}_{\parallel}}$ that cannot be decided by any BPP algorithm on more than $1/2+1/n^{\omega(1)}$ fraction of inputs.*

1. Introduction

There is a rich interplay between coding theory and computational complexity. Complexity has both benefited from and contributed to coding theory. For instance, the PCP Theorem [3, 2] uses error-correcting codes based on polynomials over finite fields while the final construction gave rise to a new kind of error-correcting code, a locally testable encoding of satisfying assignments of propositional formulas. In derandomization, error-correcting codes were (explicitly or implicitly) behind many constructions of pseudorandom generators from hard Boolean functions [19, 4, 16, 24, 23, 30]. The breakthrough extractor construction of Trevisan [25] combined good list-decodable error-correcting codes and the pseudorandom generator of Nisan and Wigderson [19]. (For many other connections between coding and complexity, see the excellent survey [27].)

Another connection between coding and complexity was observed in [26, 15], who show that direct product lemmas yield *locally, approximately list-decodable* codes. Direct product lemmas (e.g., Yao's XOR Lemma [31]) are formalizations of the intuition that it is harder to compute a function on many independent instances than on a single instance. In such lemmas, a function f that is hard to compute on some δ fraction of inputs is used to construct a function $\hat{f}$ that is hard to compute on a larger fraction (usually written as $1/2 - \epsilon$) of inputs. View $\hat{f}$ as a "coded" version of the "message" f. In an XOR lemma, it is shown how to construct a list of circuits containing a circuit that computes f with fewer than δ fraction of errors from a circuit with fewer than $1/2 - \epsilon$ fraction of errors for $\hat{f}$. This corresponds to *approximately list-decoding* the code, in that instead of exactly recovering the message f, the decoding algorithm finds a list of strings containing a string of small relative Hamming distance from f; moreover, the decoding is *local* since the constructed circuit computes an individual bit of the decoded message, not the entire message.

Approximately decodable codes are a relatively recent and potentially powerful tool in coding theory. For example, an approximately decodable code can be composed with a standard error-correcting code to boost the amount of noise that can be tolerated (see [1, 10, 11, 12] for several applications of this idea.) Weakening the notion of error-correction to include approximate error-correcting codes dramatically increases the available options for coding functions. For example, the code implicit in the XOR lemma is the *k-truncated Hadamard code*, consisting of the inner products of the message string with just those strings of Hamming weight k (where typically $k \ll N$, for N the length of the message). This code has very small sensitivity, in that flipping one bit of the input changes only a small portion of output bits, whereas any true error-correcting code has large distance and hence large sensitivity. From a combinatorial viewpoint, this means that approximately decodable codes escape many of the restrictions that have been proved for standard error-correction. In complexity terms, this allows the constructed function $\hat{f}$ to be *locally computable* from f, $\hat{f} \in \mathsf{P}^f$, an important property if we want the constructed function to have a similar complexity to the original.

As Trevisan [26] details, the standard proofs of the XOR lemma give an approximate list-decoding algorithm for the k-truncated Hadamard code running in time (and producing a list of size) *exponential* in $\mathsf{poly}(1/\epsilon)$, when given a circuit with ϵ correlation with the codeword. He observes that the list-size and hence time for such an algorithm is exponential in the amount of *advice* the construction uses. Here, we reduce the amount of advice in a proof of the XOR Lemma, giving an approximate list-decoding algorithm with polynomial time and list length for $\epsilon > 1/\mathsf{poly}(k)$ for any polynomial *poly*. However, we obtain a somewhat weaker approximation ratio δ. As a consequence, we get a strong hardness amplification result for $\mathsf{P}^{\mathsf{NP}_{\|}}$, the class of problems reducible to NP through one round of parallel oracle queries.

While interesting in themselves, our results are also significant in that they utilize the equivalence of XOR lemmas and approximate decoding at several points. As in many other cases, knowing an equivalence between problems in two different domains gives researchers leverage to make progress in both domains.

Below, we give a more precise description of our results.

1.1 Direct Product Lemma and Direct Product Codes

Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, we define its k-wise direct product as $f^k(x_1, \ldots, x_k) = f(x_1) \ldots f(x_k)$. The Direct Product Lemma [31, 18, 14, 9, 16] says that if a Boolean function f cannot be computed by any size s Boolean circuit on more than $1 - \delta$ fraction of inputs, then its direct product function f^k cannot be computed by any circuit of size $s' = s * \mathsf{poly}(\epsilon, \delta)$ on more than $\epsilon = e^{-\Omega(\delta k)}$ fraction of inputs. Viewing f^k as a "direct product" encoding of the message f, we can interpret the Direct Product Lemma as an approximate list-decoding of the code.

The known proofs of the Direct Product Lemma [14, 9, 16] are constructive: there is an algorithm that, given a circuit of size s' that computes f^k on more than ϵ fraction of inputs, constructs a Boolean circuit of size s that computes f on more that $1 - \delta$ fraction of inputs. However, in all these proofs, the algorithm constructing a circuit for f needs some nonuniform advice. Instead of producing a circuit for f, the algorithm produces a *list* of circuits one of which computes f on more than $1 - \delta$ fraction of inputs; the logarithm of the size of this list is exactly the number of bits of nonuniform advice used by the construction. In the known proofs, the advice size is $\mathsf{poly}(1/\epsilon)$ (cf. [26]) and so the size of the list of circuits is $2^{\mathsf{poly}(1/\epsilon)}$. In terms of approximate list-decoding, this is the list of possible approximate messages, one of which is δ-close to the original message. For information-theoretically optimal list decoders, this list size should not exceed $O(1/\epsilon^2)$ (see the full version of this paper), which corresponds to $O(\log(1/\epsilon))$ bits of advice.

In this paper, we achieve list size $\mathsf{poly}(1/\epsilon)$, albeit only for large $\epsilon = \Omega(\mathsf{poly}(1/k))$.

We say that a circuit C ϵ-computes the $1-\gamma$-Direct Product f^k, if, with probability at least ϵ over a random k tuple $x_1, ..x_k$ of inputs, $C(x_1, ..x_k)_i = f(x_i)$ for at least $(1-\gamma)k$ values of i. Note that this is weaker than traditional direct product notions, since C is only required to get *most* of the answers right. However, since we use this weaker notion recursively inside our main algorithm, and since it gives a stronger result, we state our main theorem for this notion of computing the direct product.

Theorem 1 (Main Theorem). *There is a randomized algorithm A with the following property. Let f be any n-variable Boolean function. Suppose a circuit C ϵ-computes $(1 - k^{-0.4})$-Direct Product f^k, where $\epsilon = \Omega(\mathsf{poly}(1/k))$. Then the algorithm A, given C, outputs with probability at least $\epsilon' = \mathsf{poly}(\epsilon)$ a circuit C'' such that C'' agrees with f*

on at least $1-\rho$ *fraction of inputs, where* $\rho = O(k^{-0.1})$. *The running time of the algorithm A, and hence also the size of* C'', *is at most* $\mathsf{poly}(|C|, 1/\epsilon)$.

Note that the above theorem implies a list size of $\mathsf{poly}(1/\epsilon)$, since by running the algorithm A $\mathsf{poly}(1/\epsilon)$ times, we can construct a list of circuits, one of which almost certainly is a $1-\rho$ approximation to f. When $\epsilon > e^{-k^{\alpha}}$, for a sufficiently small constant $\alpha > 0$, we get a randomized approximate list-decoding algorithm for k-wise direct product codes that has the running time and list size *quasi-polynomial* in $1/\epsilon$.

Combining our local approximate list-decoding algorithms with the list-decoding algorithm for Hadamard codes due to Goldreich and Levin [8], we get local approximate list-decoding algorithms for truncated Hadamard codes, whose running time and list size correspond to those of the direct-product decoding algorithms.

1.2 Uniform hardness amplification

The main application of the Direct Product Lemma (or Yao's XOR Lemma) is to hardness amplification: If a Boolean function f is somewhat hard to compute on average, its XOR function $f^{\oplus^k}(x_1,\ldots,x_k) = \oplus_{i=1}^{k} f(x_i)$ is much harder on average. The known proofs of Yao's XOR Lemma use nonuniform reductions and so they give hardness amplification only in the nonuniform setting of Boolean circuits.

Impagliazzo and Wigderson [17] consider the setting of *uniform* hardness amplification. Here, one starts with a Boolean function family that is somewhat hard on average to compute by *probabilistic polynomial-time algorithms*, and defines a new Boolean function family that is much harder on average. Ideally, one would start from a function that is hard $1/\mathsf{poly}(n)$ of the time for some fixed polynomial $\mathsf{poly}(n)$, and end with a function in the same complexity class that is hard $1/2 - 1/\mathsf{poly}(n)$ of the time, for any $\mathsf{poly}(n)$. Yao's XOR Lemma amplifies hardness of a Boolean function family f also in this setting, but only if we are given oracle access to f. This oracle access can be eliminated under certain circumstances, e.g., if the distribution $(x, f(x))$ can be sampled, or if f is downward self-reducible and random self-reducible. Impagliazzo and Wigderson [17] use this to show uniform hardness amplification for $\#\mathsf{P}$. Trevisan and Vadhan [29] show that uniform hardness amplification is also possible in PSPACE and in EXP.

Trevisan [26, 28] considers uniform hardness amplification for languages in NP; the nonuniform case was studied in [20, 13]. Trevisan [28] shows uniform amplification from $1/\mathsf{poly}(n)$ to $1/2 - 1/\mathsf{poly}\log(n)$. Note that the final hardness falls short of the desired $1/2 - 1/\mathsf{poly}(n)$. The reason for this is the use of $2^{\mathsf{poly}(1/\epsilon)}$-bit advice by the BPP algorithm that, given a circuit computing an NP language L' on more than $1/2+\epsilon$ fraction of inputs, produces a circuit computing L on more than $1 - 1/\mathsf{poly}(n)$ fraction of inputs. If $\epsilon = \log^{\alpha} n$, for sufficiently small $\alpha > 0$, then the required amount of advice is $O(\log n)$. Using the average-case version of the "search-to-decision" reduction for NP [5], this logarithmic advice can be eliminated in time $2^{O(\log n)} = \mathsf{poly}(n)$ by, essentially, trying all possible advice strings.

Using our efficient approximate list-decoding algorithm for truncated Hadamard codes, we achieve better uniform hardness amplification, but only for the class $\mathsf{P}^{\mathsf{NP}_\parallel}$. Namely, we show that if $\mathsf{P}^{\mathsf{NP}_\parallel}$ contains a language L that cannot be computed by BPP algorithms on more than $1 - 1/\mathsf{poly}(n)$ fraction of inputs, then there is another language $L' \in \mathsf{P}^{\mathsf{NP}_\parallel}$ that cannot be computed by BPP algorithms on more than $1/2 + 1/n^{\omega(1)}$ fraction of inputs. The reason we get amplification for $\mathsf{P}^{\mathsf{NP}_\parallel}$ rather than NP is our use of the XOR function as an amplifier; if $f \in \mathsf{NP}$, then its XOR function $f^{\oplus^k}(x_1,\ldots,x_k) = \oplus_{i=1}^{k} f(x_i)$ is not necessarily in NP, although it is certainly in $\mathsf{P}^{\mathsf{NP}_\parallel}$. (Achieving the same amplification for NP seems to require a similar result with a *monotone* function replacing $\oplus^k$.)

1.3 Our techniques

The proof of the Direct Product Lemma in [16] yields an efficient algorithm $LEARN$ with the following property: Given as input a "small" circuit C computing the direct product function f^k for at least ϵ fraction of inputs, *and* given about $(1/\epsilon^2)$ random samples of the form $(x, f(x))$ for independent uniformly distributed xs, the algorithm $LEARN$ produces, with high probability, a "small" circuit computing f on at least $1-\delta$ fraction of inputs, for $\delta \approx \frac{\log(1/\epsilon)}{k}$. In our case, we have a circuit C, but no labeled examples $(x, f(x))$.

Our construction combines three technical steps:

Boosting direct products We give an algorithm for converting a circuit that ϵ-computes $1-\gamma$ direct product f^k to one that ϵ'-computes $1-\gamma'$ direct product $f^{k'}$, where $\epsilon' \geq poly(\epsilon)$, $\gamma' \in O(\gamma + k^{-.4})$ and $k' = k^{1.5}$. Repeating recursively, we can go from a circuit to compute a direct product on k inputs to one that approximately computes a direct product on any polynomial in k inputs. This DP booster can be thought of as approximately list-decoding the k-truncated Hadamard code for the special case where $N \leq \mathsf{poly}(k)$.

The main idea of the DP booster is: given $k^{1.5}$ inputs, first guess one subset S of k inputs and hope that the given circuit (approximately) computes the direct product on S. Given that this first step succeeds, we use the values of f on inputs from S as a reality check on random subsets T, accepting the values for inputs in T if there are few inconsistencies with the assumed values for S. By the birthday paradox, S and T will have a large intersection, so if the values for S are (mostly) correct, we are unlikely to accept any T for which the values are not mostly correct. By combining many random consistent T's, we eventually fill in correct guesses for most of the inputs in the entire set.

Self-advising learning algorithm The advice we need for $LEARN$ is in the form of many random examples $(x, f(x))$. A circuit ϵ-computing a direct product has an ϵ chance of providing k such examples. To get enough samples, we first need to boost the direct product until $k' = poly(1/\epsilon)$. However, the resulting samples may be correlated, and our circuit for k' only computes an approximate direct product. We quantify the first problem through a *sampling lemma*, which argues that a random subset of the inputs where the direct product circuit is (approximately) successful is almost uniform.

Fault-tolerant learning algorithm Finally, we address the last problem that some of the advice may in fact be misleading, not actually being the value of the function on the example input. To handle this, we give a fault-tolerant analysis of the learning algorithm from [16], showing that the algorithm works even if a small fraction of the advice is incorrect.

Outline of the paper We describe the main tools used in our approximate list-decoding algorithm in Section 2, and sketch the proof of our Main Theorem (Theorem 1) in Section 3. Applications to uniform hardness amplification are given in Section 4. We give concluding remarks in Section 5.

2 Our tools

2.1 Extracting labeled examples from the direct product circuit

The following lemma will allow us to extract slightly imperfect labeled examples $(x, f(x))$ from a circuit approximately computing the direct product of f.

Lemma 2 (Sampling Lemma). *Let C be a circuit that ϵ-computes $(1-\gamma)$-direct product f^k, for some Boolean function $f : \{0,1\}^n \to \{0,1\}$ and parameters ϵ, γ where $\gamma \geqslant 1/k^{0.4}$. Let $\bar{x} = (x_1, \ldots, x_k) \in (\{0,1\}^n)^k$ be a uniformly random tuple, let $\bar{x'} = (x_{i_1}, \ldots, x_{i_{\sqrt{k}}})$ be a uniformly random subtuple of $\bar{x}$, and let $b_{i_1} \ldots b_{i_{\sqrt{k}}}$ be the values of $C(\bar{x})$ corresponding to the subtuple $\bar{x'}$. Then there is an event E that occurs with probability $\Omega(\epsilon)$ so that, conditioned on E occurring, the following hold:*

1. *the statistical distance between the distribution $\bar{x'}$ and the uniform distribution on $(\{0,1\}^n)^{\sqrt{k}}$ is at most $0.6\sqrt{\log(1/\epsilon)/\sqrt{k}} + e^{-\Omega(k^{0.1})}$, and*
2. *for all but at most $O(\gamma)$ fraction of elements x_{i_j}s in $\bar{x'}$, we have $b_{i_j} = f(x_{i_j})$.*

For the proof of this lemma, we need the following result implicit in [21]; the proof is given in the Appendix.

Lemma 3. *Let $G \subseteq \{0,1\}^{mn}$ be any subset of size $\epsilon 2^{mn}$. Let U be a uniform distribution on the set $\{0,1\}^n$, and let D be the distribution defined as follows: pick a tuple $(x_1, \ldots, x_m)$ of n-bit strings uniformly from the set G, pick an index i uniformly from $[m]$, and output x_i. Then the statistical distance between the distributions U and D is less than $0.6\sqrt{\frac{\log 1/\epsilon}{m}}$.*

Lemma 2 follows from Lemma 3: view a k-tuple as a $\sqrt{k}$-tuple of $\sqrt{k}$-tuples, and let G be the set of tuples for which the circuit C is approximately correct. The event E is that the tuple falls in G and that C is approximately correct on the chosen subtuple.

2.2 Direct Product amplification

The following lemma will allow us to use a circuit computing the direct product f^k in order to compute the direct product $f^{k'}$ for a larger k'.

Lemma 4 (One-step DP Amplification Lemma). *There is a randomized algorithm A with the following property. Let f be any n-variable Boolean function. Suppose a circuit C ϵ-computes $(1-\gamma)$-Direct Product f^k, where $\epsilon > e^{-k^{0.001}}$. Then the randomized algorithm A, given C, outputs with probability at least ϵ' a circuit C' ϵ'-computing $(1-\gamma')$-Direct Product $f^{k'}$, where $\epsilon' = \Omega(\epsilon^2)$, $\gamma' \leqslant O(\gamma + k^{-0.4})$, and $k' = k^{3/2}$. The running time of A, and hence also the size of C', is polynomial in the size of C and $1/\epsilon$.*

Proof. We only give a sketch of the proof here; the complete proof will be in the full paper. We will need the following definitions. Given a k'-tuple $\bar{x} = (x_1, \ldots, x'_k)$, and T a subset of $[k']$ of size k, let $\bar{x}_T$ be the restriction of $\bar{x}$ to the elements of T. If j is the i'th smallest element of T, let $b_j^T = C(x_T)_i$. We call a set T of size k α-good if $b_j^T = f(x_j)$ for all but αk locations $j \in T$. We say that a set T is *accepted w.r.t.* S if $|\{j \in S \cap T \mid b_j^S \neq b_j^T\}|/|S \cap T| < 3\gamma + k^{-0.4}$.

The algorithm is given in figure 1.

We will argue that Algorithm 1 $\Omega(\epsilon')$-computes $(1-\gamma')$-Direct Product $f^{k'}$.

```
INPUT: $\bar{x} = (x_1, \ldots, x_{k'}) \in \{0,1\}^{nk'}$
OUTPUT: $(output_1, \ldots, output_{k'}) \in \{0,1\}^{k'}$
ORACLE ACCESS: algorithm $C$ that $\epsilon$-computes $(1-\gamma)$-Direct Product $f^k$
PARAMETERS: $\epsilon > e^{-k^{0.001}}$, $\rho = 3\gamma + k^{-0.4}$, $timeout = (96k' \log k')/\epsilon$, $\alpha = 5(\gamma + k^{-0.4})$.

1. For every $i \in [k']$, set $Answers_i$ =empty string.
2. Choose $S \subseteq [k']$ of size $|S| = k$ uniformly at random.
3. Compute $C(\bar{x}|_S)$, and hence $b_j^S$ for $j \in S$.
4. Repeat lines 5–9 for at most $timeout$ times:
5.     Choose $T \subseteq [k']$ of size $|T| = k$ uniformly at random.
6.     Compute $C(\bar{x}|_T)$ and hence $b_j^T$ for $j \in T$.
7.     Let $m = |\{j \in S \cap T \mid b_j^S \neq b_j^T\}|/|S \cap T|$.
8.     If $m < \rho$,    % if T is accepted w.r.t. S
9.         then choose a random $i \in T$ and extend the string $Answers_i$ with the bit $b_i^T$.
10. For every $i \in [k']$, let $count_i = |Answers_i|$.
11. Let $total = \sum_{i \in [k']} count_i$.
12. If $count_i < \frac{total}{2k'}$, then set $output_i = r_i$ for a random bit $r_i$;
13.     else set $output_i = \mathrm{Majority} Answers_i$.
```

Algorithm 1: Direct Product amplifier

For our argument, we will need the following two claims; the proofs are given in the full version of the paper.

Claim 5. *Let $\bar{x}$ be a k'-tuple such that at least $\epsilon/2$ random subsets of $[k']$ of size k are γ-good. Conditioned on both T and S being γ-good random subsets of $[k']$,*

$$\mathbf{Pr}_{S,T}[T \textit{ is accepted w.r.t. } S] \geqslant 1 - e^{-\Omega(k^{0.1})}.$$

Claim 6. *Let $\bar{x}$ be a k'-tuple such that at least $\epsilon/2$ random subsets of $[k']$ of size k are γ-good. For at least $1 - e^{-\Omega(k^{0.1})}$ fraction of γ-good sets S, at least $1 - e^{-\Omega(k^{0.1})}$ fraction of sets T that are accepted w.r.t. S are $\alpha = 5(\gamma + k^{-0.4})$-good.*

By a Markov-style argument, for at least $\epsilon/2$ fraction of k'-tuples $\bar{x}$, there will be at least $\epsilon/2$ fraction of γ-good sets $S \subseteq [k']$ of size k. Fix such a k'-tuple $\bar{x}$. We will bound the probability that the algorithm makes many mistakes on this tuple.

By Claims 5 and 6, we know that with probability at least $(\epsilon/2)(1 - e^{-\Omega(k^{0.1})}) \geqslant \epsilon/3$, the set S chosen in line 3 of our algorithm will have the following properties:

1. S is γ-good,
2. all but $e^{-\Omega(k^{0.1})}$ fraction of γ-good sets T are accepted w.r.t. S, and
3. all but $e^{-\Omega(k^{0.1})}$ fraction of sets T accepted w.r.t. S are α-good, for $\alpha = 5(\gamma + k^{-0.4})$.

Assume that a set S satisfying properties (1)–(3) was chosen in step 2.

There are two possible causes for incorrect labels to x_i: *(1)* $count_i$ does not reach $total/2k'$ and the label is chosen at random, or *(2)* the majority of bits in $Answer_i$ are incorrect, and there are at least $total/2k'$ of them.

For the first problem, note that there are an $\Omega(\epsilon)$ fraction of sets T that are γ-good, and most of these are accepted with respect to S. So we can apply an analogue of Lemma 3 for the case of sets rather than tuples to see that the distribution on i given that T is accepted is within distance $\beta \overset{\text{def}}{=} O(\sqrt{\frac{\log(1/\epsilon)}{k}}) \leqslant O(k^{-0.4})$ of the uniform distribution over $[k']$. Thus, there is only an $O(k^{-.4})$ fraction of i's whose conditional expectation is less than $3/(4k')$, and, with high probability, all other i's have $count_i > total/(2k')$.

Secondly, with high probability, all of the chosen T which are accepted with respect to S are α-good. For each such T, there is at most an α probability of a mistake being placed in some $Answer_i$. So with high probability, there are at most $O(\alpha total)$ mistakes in all $Answer_i$'s. Each i for which the majority of $Answer_i$ is invalid and for which $count_i > total/(2k')$ is responsible for $total/(4k')$ such mistakes, meaning there can be at most $4\alpha k'$ such i's in all.

This gives a total of $O(\alpha + k^{-.4})$ mistakes, with high probability. □

For every constant c, a repeated application of Lemma 4 for a constant number of times yields with probability $\epsilon' = \mathsf{poly}(\epsilon)$ a circuit C' ϵ'-computing $(1-\gamma')$-Direct Product $f^{k'}$, where $\epsilon' = \mathsf{poly}(\epsilon)$, $\gamma' \leqslant O(\gamma + k^{-0.4})$, and $k' = k^c$. The running time of this amplification algorithm, and hence also the size of C', is polynomial in the size of C and $1/\epsilon$.

2.3 Direct products with faulty advice

We recall the proof of the Direct Product Lemma from [16]. Given a circuit C that computes the direct product function f^k with probability at least ϵ, consider the following distribution $\mathcal{F}$ on randomized circuits F. On input x, pick $i \in [k]$ uniformly at random, and set $x_i = x$. For each $j \in [k] \setminus \{i\}$, get a sample $(x_j, f(x_j))$ where x_j is uniformly distributed. Evaluate the circuit C on the input $(x_1, \ldots, x_k)$. Let z be the number of indices $j \in [k] \setminus \{i\}$ where the jth output of the circuit C disagrees with the value $f(x_j)$. With probability 2^{-z}, output the ith output of the circuit C, and with the remaining probability $1 - 2^{-z}$ output a random coin flip.

Impagliazzo and Wigderson [16] argue that, for every subset H of at least δ fraction of inputs, a random circuit sampled according to $\mathcal{F}$ will compute f on a random input from H with probability at least $1/2 + \Omega(\epsilon)$. Then they conclude that the circuit obtained by applying the majority function to a small number of sampled circuits from $\mathcal{F}$ will compute f on all but at most δ fraction of inputs.

We will generalize the argument of [16] in two ways. First, in the definition of the probability distribution $\mathcal{F}$, instead of sampling $(k-1)$-tuples $(x_j, f(x_j))$ for uniformly distributed n-bit strings x_j's, we will use the x_j's that come from a probability distribution D over $\{0,1\}^{n(k-1)}$ that is only *statistically close* to uniform. Secondly, we assume that rather than getting the correct value $f(x_j)$ for all $k-1$ strings x_j in the sampled $(k-1)$-tuple from D, we get the correct value $f(x_j)$ for *most* x_j's in the tuple. We will show that even with these imperfect samples, we can construct a circuit computing the function f on at least $1-\rho$ fraction of inputs, for some $\rho \approx \delta$.

The following lemma can be proved by appropriately modifying the arguments in [16]; a complete proof is given in the full paper.

Lemma 7. *Let f be an n-variable Boolean function. Suppose a circuit C ϵ-computes f^k, for some $\epsilon > e^{-k^{0.001}}$. Let D be a distribution on $(k-1)$-tuples $(x_1, b_1), \dots, (x_{k-1}, b_{k-1})$ such that (i) the x_js are independent uniformly distributed random variables over $\{0,1\}^n$, and (ii) $b_j = f(x_j)$ for at least $(1-\gamma)$ fraction of $j \in [k-1]$, where $\gamma \leqslant k^{-0.3}$. Then there is a probability distribution $\mathcal{F}$ over randomized Boolean circuits F such that, for every set $H \subseteq \{0,1\}^n$ of density $\delta \geqslant \sqrt{\gamma} + \frac{8\ln 100/\epsilon}{k}$,*

$$\mathbf{Pr}_{F \leftarrow \mathcal{F}, x \leftarrow H}[F(x) = f(x)] > 1/2 + \Omega(\epsilon).$$

Moreover, $\mathcal{F}$ is sampleable in time $\mathsf{poly}(1/\epsilon, |C|)$ given input C and one sample from D.

Using the lemma above, we can use the arguments of [16] to obtain a learning algorithm $LEARN$ that, given access to random samples from distribution D, will produce with high probability a circuit that computes f on all but δ fraction of inputs.

Theorem 8. *Let f, C, D, and δ be as in the statement of Lemma 7. There is a randomized algorithm $LEARN$ that, given a circuit C and oracle access to the distribution D, outputs with high probability a circuit C' that computes f on at least $1-\delta$ fraction of inputs. The running time of the algorithm $LEARN$, and hence also the size of C', is at most $\mathsf{poly}(|C|, 1/\epsilon)$.*

Proof. By Lemma 7, we get, for sufficiently small constant $\alpha > 0$, that the set $Bad = \{x \in \{0,1\}^n \mid \mathbf{Pr}_{F \leftarrow \mathcal{F}}[F(x) = f(x)] \leqslant 1/2 + \alpha\epsilon\}$ must have density less than δ. Thus, by Chernoff, the majority of $O(n/\epsilon^2)$ circuits sampled from $\mathcal{F}$ will be correct on every input $x \notin Bad$ with probability greater than $1 - 2^{-2n}$. By a Markov-style argument, we get that for more than $1-2^{-n}$ fraction of the majority circuits, each such circuit is correct on more than $1-2^{-n}$ fraction of inputs $x \notin Bad$. It follows that if we pick one such majority circuit at random, then with high probability it will be correct on all inputs $x \notin Bad$, and so it will compute the function f on at least $1-\delta$ fraction of inputs. □

The following lemma shows that this algorithm $LEARN$ continues to work even when given oracle access to a slightly corrupted oracle D where the distribution on x_js is not uniform but rather only statistically close to uniform.

Lemma 9. *Let $LEARN$ be the algorithm guaranteed to exist by Theorem 8. Let f be an n-variable Boolean function. Suppose a circuit C ϵ-computes f^k, for some $\epsilon > e^{-k^{0.001}}$. Let D be a probability distribution over $(\{0,1\}^{(n+1)(k-1)})^{O(n/\epsilon^2)}$, viewed as $O(n/\epsilon^2)$ blocks of $(k-1)$-tuples of pairs (x_j, b_j) where $x_j \in \{0,1\}^n$ and $b_j \in \{0,1\}$, such that: (1) for every sample from D, each block $(x_1, b_1), \dots, (x_{k-1}, b_{k-1})$ of the sample is such that $b_j = f(x_j)$ for at least $(1-\gamma)$ fraction of $j \in [k-1]$, for some $\gamma \leqslant k^{-0.3}$, and (2) for the distribution D' obtained from D by deleting the bits b_i and outputting the strings x_i, the statistical distance between D' and the uniform distribution over $(\{0,1\}^{n(k-1)})^{O(n/\epsilon^2)}$ is at most κ. Then, on input C, the algorithm $LEARN$, replacing each oracle call to the distribution by a block from a single sample from D, will, with probability at least $1 - 2^{-n} - \kappa$, output a circuit C' that computes f on at least a $1-\rho$ fraction of inputs, for $\rho = O(\frac{\log(1/\epsilon)}{k} + \sqrt{\gamma})$.*

Proof. View the distribution D as the pair of random variables (D', D''), where D'' is the distribution on bits b_i, conditioned on a given sample from D'. Assume that the conclusion of the lemma failed. Then the following is a statistical test distinguishing between D' and the uniform distribution: Given a sample a of $O(n/\epsilon^2)$ blocks of $(k-1)$-tuples $x_1, \dots, x_{k-1}$, sample from D'' conditioned on the given sample a. (If the sample a has zero probability under D', then we know that it came from the uniform distribution. So we may assume, without loss of generality, that a has nonzero probability in D'.) Then run the algorithm described above that constructs a circuit for the function f. If the constructed circuit agrees with f on at least $1-\rho$ fraction of inputs, then accept; otherwise reject.

By the preceding arguments, we know that when the sample a comes from the uniform distribution, with probability at least $1-2^{-n}$ the constructed circuit computes f on at least $1-\rho$ fraction of inputs, and hence our statistical test accepts with probability at least $1-2^{-n}$. On the other hand, if a comes from D', then, by the assumption, the constructed circuit is good with probability less than $1-2^{-n}-\kappa$, and so the statistical test accepts with probability less than $1-2^{-n}-\kappa$. This implies that the described statistical test κ-distinguishes D' from the uniform distribution. □

```
INPUT: $n, t, k \in \mathbb{N}$.
OUTPUT: $t$ blocks of $(k-1)$-tuples of the form $(x_1, b_1) \dots (x_{k-1}, b_{k-1})$, where $x_i \in \{0,1\}^n$, $b_i \in \{0,1\}$.
ORACLE ACCESS: algorithm $C$ that $\epsilon$-computes Direct Product $f^k$.
PARAMETERS: $\epsilon = \Omega(\mathsf{poly}(1/k))$, $k = \mathsf{poly}(n)$, $t = O(n/\epsilon^2)$.

1. Set $k' = (t(k-1))^2$.
2. Run the Direct Product Amplification algorithm (Algorithm 1) from Lemma 4 on the circuit $C$ with this value of $k'$ (for a constant number of times), obtaining as an output a circuit $C'$.
3. Run the circuit $C'$ on a random $k'$-tuple of $n$-bit strings.
4. Select $t(k-1)$ strings $x_1, \dots, x_{t(k-1)}$ in this $k'$-tuple uniformly at random, and randomly partition these strings into $t$ blocks of $(k-1)$-tuples.
5. Output the $t$ blocks of $(k-1)$-tuples of the form $(x_1, b_1) \dots (x_{k-1}, b_{k-1})$ where $b_j$ is the output of $C'$ corresponding to the input string $x_j$.
```

Algorithm 2: Sampler

3 Approximate List-Decoding of the Direct Product Code: Proof of Main Theorem

In this section we sketch the proof of our Main Theorem (Theorem 1); a complete proof is given in the full version of the paper.

Recall that in order to apply Lemma 9, we need to have access to a probability distribution D on $t = O(n/\epsilon^2)$ blocks of $(k-1)$-tuples of the form $(x_1, b_1) \dots (x_{k-1}, b_{k-1})$, where the tuple of x_is is statistically close to the uniform distribution and, in each block, $b_j = f(x_j)$ for at least $1-\gamma$ fraction of positions, for some $\gamma \leqslant k^{-0.3}$. Given a circuit C that ϵ computes f^k, we sample advice using Algorithm Sampler given in Figure 2.

We claim that, conditioning on an event that occurs with probability at least $\mathsf{poly}(\epsilon)$, the output produced by the sampling algorithm $Sampler$ is distributed according to a distribution D with the requisite properties. Indeed, first note that by our assumption about ϵ and k, we have that $k' = O(\mathsf{poly}(k))$. By repeated application of Lemma 4, with probability at least $\epsilon' = \mathsf{poly}(\epsilon)$, the circuit C' in Step 2 of the sampling algorithm ϵ'-computes $(1-\gamma)$-Direct Product $f^{k'}$ for $\gamma \leqslant O(k^{-0.4}) \leqslant k^{-0.3}$. Let E' denote the event that such a good circuit C' is produced in Step 2 of the sampling algorithm. Then, conditioned on E', we have by the Sampling Lemma (2) an event E that occurs with probability $\Omega(\epsilon')$ so that conditioned on E:

- For a random subset $T \subseteq [k']$ of size $t(k-1)$, the distribution on $t(k-1)$-tuples $\bar{x}|_T$, obtained from $\bar{x}$ by restricting it to the indices in T, has statistical distance at most $\kappa = 0.1$ from the uniform distribution on all $t(k-1)$-tuples from $\{0,1\}^{nt(k-1)}$.
- The fraction of x_i, $i \in T$, with $b_i \neq f(x_i)$ is at most $O(\gamma)$.

By Chernoff, with high probability, the last condition also occurs for each block obtained by a random partition in Step 4 of the sampling algorithm; let E'' be the event that Step 4 produced such a good partition. Let D be the distribution produced by Sampler, conditioned on events E', E, and E''.

Finally, given an output from the algorithm $Sampler$, we apply the algorithm of Lemma 9 and obtain a circuit C'' satisfying the following. Conditioned on $Sampler$ producing a sample from D, with probability at least $1 - 2^{-n} - 2\kappa \geqslant 1/2$ over the internal random coin tosses of the algorithm in Lemma 9, the constructed circuit C'' computes f on at least $1-\rho$ fraction of inputs. Lifting the conditioning on $Sampler$, we get that the probability of producing a circuit C'' that computes f on at least $1-\rho$ fraction of inputs is at least $\mathsf{poly}(\epsilon)$, as required.

When $\epsilon > e^{-k^{0.001}}$, we get a version of Theorem 1 where the list size and the running time of A are quasi-polynomial in $1/\epsilon$; we omit the details.

4 Applications

4.1 Local approximate list-decoding of truncated Hadamard codes

For $n, k \in \mathbb{N}$, a k-truncated Hadamard encoding of a message $msg \in \{0,1\}^n$ is defined as a string $code_{msg} \in \{0,1\}^{\binom{n}{k}}$, where the codeword is indexed by k-sets $s \subseteq [n]$, $|s| = k$, and $code_{msg}(s) = \oplus_{i \in s} msg(i)$. Using our local approximate list-decoding algorithm for direct product codes in Theorem 1 and the list-decoding algorithm for Hadamard codes of Goldreich and Levin [8], we get an efficient approximate list-decoding algorithm for k-truncated Hadamard codes. Equivalently, one can think of this as an advice-efficient proof of the Yao XOR Lemma.

Lemma 10 (Advice-efficient XOR Lemma). *There is a randomized algorithm A with the following property. Let f be any n-variable Boolean function. Suppose a circuit C computes the function $f^{\oplus^k}(x_1, \dots, x_k)$ on at least $1/2 + \epsilon$ fraction of inputs, where $\epsilon = \Omega(\mathsf{poly}(1/k))$. Then the algorithm A, given C, outputs with probability at least $\epsilon' = \mathsf{poly}(\epsilon)$ a circuit C' such that C' agrees with f on at least $1-\rho$ fraction of inputs, where $\rho = O(k^{-0.1})$. The running time of the algorithm A, and hence also the size of C', is at most* $\mathsf{poly}(|C|, 1/\epsilon)$.

Proof. We first use the result of [8] to convert the circuit C to a circuit $\overline{C}$ that $poly(\epsilon)$ computes f^{2k}, and then apply Theorem 1 to $\overline{C}$. The algorithm in [8] can be viewed as an algorithm for list-decoding the Hadamard code.

Lemma 11 ([8]). *There is a probabilistic algorithm A with the following property. Let $s \in \{0,1\}^n$ be any string, and let $B : \{0,1\}^n \to \{0,1\}$ be any predicate such that $\mathbf{Pr}_{r \in \{0,1\}^n}[B(r) = \langle s, r\rangle] \geqslant 1/2 + \gamma$, for some $\gamma > 0$. Then, given oracle access to B, the algorithm A runs in time $\mathsf{poly}(n, 1/\gamma)$, and outputs the string s with probability at least $\Omega(\gamma^2)$.*

Consider a random tuple $S = \langle x_1, \ldots, x_{2k}\rangle$ of inputs. Let s be the vector of bits $f(x_i)$. With probability at least $\epsilon/2$, the conditional probability that C computes $f^{\oplus^k}$ for a random subtuple T of size k in S is at least $\epsilon/2$. Then, given a random bit vector r of length $2k$, if the Hamming weight of r is k, let B simulate C on the sub-tuple T_r of those x_i's where $r_i = 1$; otherwise, B outputs a random bit. Then B has correlation $O(\epsilon/\sqrt{k})$ of predicting $\langle r, s\rangle$. So applying the Goldreich-Levin algorithm A with oracle B gives, with probability $\Omega(\epsilon^2/k)$, the string s. Thus, the combination $\overline{C} = A^B$ is a circuit that $\Omega(\epsilon^3/k)$ computes f^{2k}.

We then apply Theorem 1 to $\overline{C}$. □

This immediately gives us a polynomial-time approximate list-decoding of the k-truncated Hadamard code.

Theorem 12. *For any polynomial p, there is a randomized algorithm A with the following properties. Let msg be any N-bit string, and let $code_{msg}$ be its k truncated Hadamard code. Let B be any predicate of nk bit inputs, so that $B[v] = (code_{msg})_v$ for at least $1/2+\epsilon$ fraction of positions v, where $\epsilon \geq p(1/k)$. Then the algorithm A, given oracle B, outputs with high probability a list of at most $\mathsf{poly}(1/\epsilon)$ strings m' such that there is at least one m' on the list that agrees with msg in at least $1-\rho$ fraction of positions, where $\rho = O(k^{-0.1})$. The running time of the algorithm A is at most $\mathsf{poly}(N, 1/\epsilon)$.*

4.2 Uniform hardness amplification in $\mathsf{P}^{\mathsf{NP}_{\|}}$

We say that a Boolean function family f is δ-hard with respect to probabilistic polynomial-time algorithms if any such algorithm computes f correctly on at most $1-\delta$ fraction of n-bit inputs, where δ is some function of the input size n. Similarly we can define hardness with respect to probabilistic polynomial-time algorithms using advice. We use the model of probabilistic algorithms taking advice as defined in [29]: the advice may depend on the internal randomness of the algorithm but is independent of the given input.

Lemma 10 immediately gives us hardness amplification for probabilistic algorithms with small amount of advice.

Lemma 13. *Suppose $f : \{0,1\}^n \to \{0,1\}$ is Boolean function family such that, for some constant c, f is $1/n^c$-hard with respect to any probabilistic polynomial-time algorithm with $O(\log n)$-size advice. Then the function $f^{\oplus^k}$, for $k = n^{11c}$, cannot be $(1/2 + 1/n^d)$-computed by any probabilistic polynomial-time algorithm for any d.*

Proof. Given a probabilistic algorithm $(1/2+\epsilon)$-computing the XOR function $f^{\oplus^k}$, we apply the algorithm of Lemma 10 to it $\mathsf{poly}(1/\epsilon)$ times. This gives us a list of $\mathsf{poly}(1/\epsilon)$ Boolean circuits such that, with probability exponentially close to 1, at least one of the circuits on the list $(1-\delta)$-computes f. The advice string of size $\log \mathsf{poly}(1/\epsilon) = O(\log 1/\epsilon)$ can then be used to identify the correct circuit on the list. □

Logarithmic advice can sometimes be eliminated. For functions in NP, we can use the average-case search-to-decision reduction due to Ben-David, Chor, Goldreich, and Luby [5] to obtain the following lemma, also used by [28].

Lemma 14. *Suppose there is a language $L \in \mathsf{NP}$ and a constant c such that L is $1/n^c$-hard with respect to probabilistic polynomial-time algorithms. Then there is a language $L' \in \mathsf{NP}$ and a constant d such that L' is $1/n^d$-hard with respect to probabilistic polynomial-time algorithm taking $O(\log n)$ bits of advice.*

Combining Lemma 13 and Lemma 14, we obtain the following.

Theorem 15. *Suppose there is a Boolean function family $f \in \mathsf{NP}$ and a constant c such that f is $1/n^c$-hard with respect to probabilistic polynomial-time algorithms. Then there is a Boolean function family $g \in \mathsf{P}^{\mathsf{NP}_{\|}}$ that cannot be computed by any probabilistic polynomial-time algorithm on more that $1/2+1/n^d$ fraction of inputs, for any constant d.*

Finally, we observe that the existence of a hard function in $\mathsf{P}^{\mathsf{NP}_{\|}}$ implies the existence of a hard function in NP, and so Theorem 15 can be used to achieve uniform hardness amplification in $\mathsf{P}^{\mathsf{NP}_{\|}}$.

Theorem 16. *Suppose there is a Boolean function family $f \in \mathsf{P}^{\mathsf{NP}_{\|}}$ and a constant c such that f is $1/n^c$-hard with respect to probabilistic polynomial-time algorithms. Then there is a Boolean function family $g \in \mathsf{P}^{\mathsf{NP}_{\|}}$ that cannot be computed by any probabilistic polynomial-time algorithm on more that $1/2+1/n^d$ fraction of inputs, for any constant d.*

The proof of the above Theorem is given in the full version of the paper.

Trevisan [28] gives uniform hardness amplification for NP: If NP contains a language that is $1/\mathsf{poly}(n)$-hard with respect to probabilistic polynomial-time algorithms, then NP contains a language that is $(1/2 - 1/\log^{\alpha} n)$-hard, for some constant α. Our Theorem 16 achieves much better hardness amplification: from $1/\mathsf{poly}(n)$-hardness to $(1/2 - 1/n^d)$-hardness for any d. However, it only applies to the class $\mathsf{P}^{\mathsf{NP}_{\parallel}}$ rather than NP.

We observe that it is also possible to prove a "scaled-up" version of Theorem 16 where the hardness is against $2^{n^{o(1)}}$-time probabilistic algorithms, and the amplification goes from hardness $1/n^c$ to $1/2 - 2^{-n^{o(1)}}$.

5 Concluding remarks

The direct product is a generic construction widely used in complexity theory and coding theory. For instance, Alon et el. [1] used (a derandomized version of) the direct product construction to obtain error-correcting codes with large distance from codes with small distance. The simplicity of this purely combinatorial construction allowed for the subsequent development of linear-time encodable and list-decodable codes [12]; this kind of efficiency appears to be impossible to achieve using algebra-based codes. The direct product construction on graphs was used by Reingold, Vadhan, and Wigderson [22] to obtain large expander graphs from smaller ones; this simple graph operation was then combined with an ingenious method to reduce the degree of the new graph while preserving its expansion properties, leading to a beautiful combinatorial construction of arbitrarily large expanders. A derandomized direct product construction similar to that of [1] was used in a recent breakthrough result of Dinur [7] as a way to amplify the "unsatisfiability" of unsatisfiable cnf formulas, which led to a significantly simpler new proof of the famous PCP Theorem. Again, the transparent combinatorial nature of the direct product construction played a major role in getting this simple proof.

In this paper, motivated by the complexity-theoretic applications to hardness amplification, we have studied the direct product construction as a tool for obtaining locally approximately list-decodable error-correcting codes. We improved the previously known decoding algorithms, achieving optimal running time and list size for decoding from a corrupted received word when the amount of corruption is less than $1 - \epsilon$ for a "large" ϵ. As an immediate application, this gives a strong average-case hardness amplification for $\mathsf{P}^{\mathsf{NP}_{\parallel}}$ with respect to BPP algorithms.

We hope that our efficient list-decoding algorithm for the direct product codes will also be useful for getting strong hardness amplification within NP. To this end, it would suffice to construct an efficiently approximately list-decodable *monotone* binary code where every bit of the codeword is a monotone Boolean function of the message, and to concatenate our direct product code with such a monotone code. This direction will be pursued in the future work.

Acknowledgements This paper was inspired by a conversation that the the first author had with Luca Trevisan at IAS in the Spring of 2005. We want to thank Luca for his suggestions that were the starting point of this research. We also want to thank Ronen Shaltiel for pointing out to us that the methods in [21] can be used in the proof of our Sampling Lemma (Lemma 2). The third author also thanks Josh Buresh-Oppenheim and Rahul Santhanam for interesting discussions. Finally, we also thank Dieter van Melkebeek for his comments, and Antonina Kolokolova for her help. The first two authors were partially supported by NSF grants 0515332 and 0313241. Views expressed are the authors' and are not endorsed by the NSF.

References

[1] 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*, 38:509–516, 1992.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the Association for Computing Machinery*, 45(3):501–555, 1998. (preliminary version in FOCS'92).

[3] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the Association for Computing Machinery*, 45(1):70–122, 1998. (preliminary version in FOCS'92).

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

[5] 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.

[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[7] I. Dinur. The PCP theorem by gap amplification. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 241–250, 2006.

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

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

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

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

[12] V. Guruswami and P. Indyk. Linear-time encodable and list decodable codes. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 126–135, 2003.

[13] A. Healy, S. Vadhan, and E. Viola. Using nondeterminism to amplify hardness. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 192–201, 2004.

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

[15] R. Impagliazzo. Hardness as randomness: A survey of universal derandomization. In *Proceedings of the ICM*, volume 3, pages 659–672, 2002. (available online at arxiv.org/abs/cs.CC/0304040).

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

[17] R. Impagliazzo and A. Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001. (preliminary version in FOCS'98).

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

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

[20] R. O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004. (preliminary version in STOC'02).

[21] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

[22] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1):157–187, 2002.

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

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

[25] L. Trevisan. Extractors and pseudorandom generators. *Journal of the Association for Computing Machinery*, 48(4):860–879, 2001. (preliminary version in STOC'99).

[26] L. Trevisan. List-decoding using the XOR lemma. In *Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science*, pages 126–135, 2003.

[27] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[28] L. Trevisan. On uniform amplification of hardness in NP. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 31–38, 2005.

[29] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity*, pages 103–112, 2002.

[30] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003. (preliminary version in STOC'02).

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

A Proof of Lemma 3

For the proof, we will use the following result from information theory which relates statistical distance from the uniform distribution and the entropy deficiency; see, e.g., [6, Lemma 12.6.1].

Lemma 17. *Let P be any probability distribution over $\{0,1\}^n$ and let U be the uniform distribution over $\{0,1\}^n$. Then $\|P-U\|_1^2 \leqslant (2\ln 2)(n-H(P))$, where H is the Shannon entropy.*

Proof of Lemma 3. Let $\bar{X} = (X_1,\dots,X_m)$ be random variables drawn according to the uniform distribution over the set G. Let $r \in [m]$ be a uniformly distributed random variable. Consider the random variable $X = X_r$. We will argue that the distribution of X is statistically close to the uniform distribution U.

First, observe that $X = X_r$ is distributed according to the average of the distributions of $X_1,\dots,X_m$, i.e., $\frac{1}{m}\sum_{i=1}^m X_i$. Thus, $H(X) = H(\frac{1}{m}\sum_{i=1}^m X_i)$. By the concavity of the entropy function, we obtain

$$H(X) \geqslant \frac{1}{m}\sum_{i=1}^m H(X_i).$$

By the independence bound on entropy, the sum of entropies is lowerbounded by the joint entropy, and so we get

$$H(X) \geqslant \frac{1}{m} H(X_1,\dots,X_m).$$

Since $\bar{X} = (X_1,\dots,X_m)$ is uniformly distributed over G, its Shannon entropy is exactly $\log_2 \frac{1}{|G|} = mn - \log_2(1/\epsilon)$. Combining this with the last lower bound on $H(X)$, we get

$$H(X) \geqslant n - \frac{\log_2 \epsilon^{-1}}{m}.$$

Finally, we use Lemma 17 above to conclude that

$$\|X-U\|_1 \leqslant \sqrt{(2\ln 2)\frac{\log_2 \epsilon^{-1}}{m}}.$$

Since the statistical distance between X and U is half of the ℓ_1 norm above, we get the claimed bound. □