Approximate List-Decoding of 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.ca

Abstract

Given a message $msg \in \{0,1\}^N$, its k-wise direct product encoding is the sequence of k-tuples $(msg(i_1),\ldots,msg(i_k))$ over all possible k-tuples of indices $(i_1,\ldots,i_k) \in \{1,\ldots,N\}^k$. We give an efficient randomized algorithm for approximate local list-decoding *of direct product codes. That is, given oracle access to a word which agrees with a k-wise direct product encoding of some message $msg \in \{0,1\}^N$ in at least $\epsilon \geqslant \mathrm{poly}(1/k)$ fraction of positions, our algorithm outputs a list of $\mathrm{poly}(1/\epsilon)$ strings that contains at least one string msg' which is equal to msg in all but at most $k^{-\Omega(1)}$ fraction of positions. The decoding is* local *in that our algorithm outputs a list of Boolean circuits so that the jth bit of the ith output string can be computed by running the ith circuit on input j. The running time of the algorithm is polynomial in $\log N$ and $1/\epsilon$. In general, when $\epsilon > e^{-k^\alpha}$ for a sufficiently small constant $\alpha > 0$, we get a randomized approximate list-decoding algorithm that runs in time quasipolynomial in $1/\epsilon$, i.e., $(1/\epsilon)^{\mathrm{poly}\log 1/\epsilon}$.*

As an application of our decoding algorithm, 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 that $1/2 + 1/n^{\omega(1)}$ 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 [AS98, ALM+98] uses error-correcting codes based on polynomials over finite fields, while the final construction gives rise to a new kind of error-correcting code, a locally testable encoding of satisfying assignments of propositional formulas. In derandomization, error-correcting codes are (explicitly or implicitly) behind many constructions of pseudorandom generators from hard Boolean functions [NW94, BFNW93, IW97, STV01, SU01, Uma03]. The breakthrough extractor

construction of Trevisan [Tre01] combines good list-decodable error-correcting codes and the pseudorandom generator of Nisan and Wigderson [NW94]. (For many other connections between coding and complexity, see the excellent survey [Tre04].)

Approximately decodable codes, where the notion of decoding is weakened to allow a small number of incorrect symbols in the decoded word, 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 [ABN$^+$92, GI01, GI02, GI03] for several applications of this idea.) Allowing both *approximate* error-correction and *list-decodability*, where the result of decoding is a small list of words that contains a word of small Hamming distance from the correct message, also dramatically increases the available options for coding functions. For instance, Trevisan [Tre03] and Impagliazzo [Imp02] observe that complexity-theoretic direct product lemmas yield *locally, approximately list-decodable* codes. We explain this connection next.

Direct product lemmas (e.g., Yao's XOR Lemma [Yao82]) 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 Boolean function f that is hard to compute on some δ fraction of inputs is used to construct a Boolean 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.

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: flipping one bit of the input changes only a small portion of output bits. On the other hand, any true error-correcting code has large distance and hence large sensitivity.

This difference means that the k-truncated Hadamard code is combinatorially *not list-decodable* for k much smaller than the length N of the message. That is, in general, there exist words w with too many codewords within a small Hamming distance from w. This is in contrast to the perfect list-decodability (with a known efficient algorithm) of the standard Hadamard code where there is no restriction on k [GL89].

However, the k-truncated Hadamard code is combinatorially *approximately list-decodable*. That is, when the fraction of corrupted symbols in the encoding of the message is less than $1/2 - \epsilon$, there will exist a small list of $O(1/\epsilon^2)$ words such that the Hamming distance δ between the correct message and the closest word on the list is at most $(\ln 1/\epsilon)/k$; see the Appendix for the precise statements and proofs.

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.

The approximate list-decodability of k-truncated Hadamard code for the parameters mentioned above (and proved in the Appendix) is known only combinatorially, and not algorithmically. It is an open problem to find an efficient algorithm that approximately list-decodes the k-truncated Hadamard code for the same parameters. As Trevisan [Tre03] details, the known 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 $\text{poly}(1/\epsilon)$, 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.

In the present paper, we reduce the amount of advice in a proof of the XOR Lemma, giving an approximate list-decoding algorithm with polynomial time and list size for $\epsilon > 1/\text{poly}(k)$ for any polynomial poly, at the price of a somewhat weaker approximation ratio δ. As a consequence, we get a strong hardness amplification result for $\mathsf{P}^{\mathsf{NP}_\parallel}$, 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.

There are three main ideas used in the construction:

Self-advising direct products: In a proof of a direct product theorem, one is converting a circuit that solves many instances of a function a small fraction of the time into one that solves a single instance a large fraction of the time. In the non-uniform setting, the converting algorithm is allowed access to many bits of advice, which in most proofs consists of random solved instances of the problem. We use the circuit solving many instances of the function on a random tuple of instances, and hope that it provides correct answers for those instances. If so, we use many of these answers in place of advice. We use a *sampling lemma* to show that these are still almost uniform, even conditioned on the success of the direct product circuit. Unfortunately, while this provides some of the required advice, it is usually not enough.

Direct product boosting: To increase the amount of advice yielded by the method above, we show how to boost a direct product circuit, constructing a circuit that approximately solves a larger direct product from one that (approximately) solves a smaller direct product. We apply this method recursively to stretch the direct product by any polynomial amount. Thus, we get a circuit that solves almost all of a larger number of instances a small fraction of the time, from one that solves all of a smaller number of instances. This can be viewed as the special case of approximately list decoding the truncated Hadamard code, when the message size and k are polynomially related.

Fault tolerant direct products: Combining the two ideas above, we can generate a large amount of slightly flawed advice, in that the instances are close to uniform and most of the supposed function values are correct. We show that at least one of the standard proofs of the direct product theorem, the one from [IW97] can tolerate such faulty advice.

Below, we describe our results and techniques more precisely.

1.1 Direct Product Lemma and Direct Product Codes

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, 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 [Yao82, Lev87, Imp95, GNW95, IW97] 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 $s' = s * \text{poly}(\epsilon, \delta)$ 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 (contrapositive to the) Direct Product Lemma as a statement about approximate list-decodability of this direct-product code.

The known proofs of the Direct Product Lemma [Imp95, GNW95, IW97] 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 actually 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 $\mathrm{poly}(1/\epsilon)$ (cf. [Tre03]) and so the size of the list of circuits is $2^{\mathrm{poly}(1/\epsilon)}$. 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 combinatorially optimal approximate list decoders, this list size does not exceed $O(1/\epsilon^2)$ (see the Appendix), which corresponds to $O(\log(1/\epsilon))$ bits of advice.

In this paper, we achieve list size $\mathrm{poly}(1/\epsilon)$, albeit only for large $\epsilon = \Omega(\mathrm{poly}(1/k))$. To state our main theorem, we need the following definition. 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 $x_1, \ldots, x_k$ 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 the traditional definition of computing a direct product where one has $\gamma = 0$. In our case 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, and let C be a circuit that ϵ-computes $(1-k^{-\mu})$-Direct Product f^k, where $\mu > 0$ and $\epsilon > e^{-k^{\alpha}}$ for a sufficiently small constant α dependent on μ (e.g., $\alpha = \min\{\mu/1000, 0.0001\}$). Given circuit C, algorithm A outputs with probability at least $\epsilon' = \epsilon^{\mathrm{poly}\log_k 1/\epsilon}$ a circuit C' such that C' agrees with f on at least $1-\rho$ fraction of inputs, where $\rho = O(k^{-\nu})$ for some constant $\nu > 0$ dependent on μ (with $\nu \approx \min\{\mu/2, 0.2\}$). The running time of the algorithm A is at most $(|C|/\epsilon)^{\mathrm{poly}\log_k 1/\epsilon}$, and the size of C' is at most $\mathrm{poly}(|C|/\epsilon)$.*

The above theorem implies an approximate list-decoding algorithm for f: by running algorithm A for $O(1/\epsilon')$ times, we get, with a constant probability, a list of circuits one of which is a $1-\rho$ approximation to f. In general, the running time of this algorithm and the size of the produced list of circuits will be at most $(1/\epsilon)^{\mathrm{poly}\log 1/\epsilon}$, i.e., *quasi-polynomial* in $1/\epsilon$. However, for the important special case where $\epsilon > \mathrm{poly}(1/k)$, the expression for ϵ' above simplifies to $\mathrm{poly}(\epsilon)$, and so we get an approximate list-decoding algorithm with running time and list size at most *polynomial* in $1/\epsilon$.

Combining our local approximate list-decoding algorithms with the list-decoding algorithm for Hadamard codes due to Goldreich and Levin [GL89], we get local approximate list-decoding algorithms for truncated Hadamard codes, whose running time and list size are essentially those of the approximate direct-product decoding algorithms.

1.2 Our techniques

The proof of the Direct Product Lemma in [IW97] 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 $\epsilon' \geq \text{poly}(\epsilon)$, $\gamma' \in O(\gamma + k^{-.4})$ 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 direct product booster can be thought of as approximately list-decoding the k-truncated Hadamard code for the special case of "large" k where $k \geqslant N^{\Omega(1)}$.

The main idea of the direct product 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 $k' = \text{poly}(1/\epsilon)$. 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 [IW97], showing that the algorithm works even if a small fraction of the advice is incorrect.

1.3 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 [IW01] 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 $1/\text{poly}(n)$ of the time for some fixed polynomial $\text{poly}(n)$, and end with a function in the same complexity class that is hard $1/2 - 1/\text{poly}'(n)$ of the time, for any polynomial $\text{poly}'(n)$. 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 [IW01] use this to show uniform hardness amplification for #P. Trevisan and Vadhan [TV02] show that uniform hardness amplification is also possible in PSPACE and in EXP.

Trevisan [Tre03, Tre05] considers uniform hardness amplification for languages in NP; the nonuniform case was studied in [O'D04, HVV04]. Trevisan [Tre05] shows uniform amplification from $1/\mathrm{poly}(n)$ to $1/2 - 1/\mathrm{poly}\log(n)$. Note that the final hardness falls short of the desired $1/2 - 1/\mathrm{poly}(n)$. The reason for this is the use of $\mathrm{poly}(1/\epsilon)$-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 $1 - 1/\mathrm{poly}(n)$ 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 [BDCGL92], this logarithmic advice can be eliminated in time $2^{O(\log n)} = \mathrm{poly}(n)$ 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 prove the following.

Theorem 2. *Suppose there is a Boolean function family $f \in \mathsf{P}^{\mathsf{NP}_\parallel}$ and a constant c such that f cannot be computed by any probabilistic polynomial-time algorithm on more than $1 - 1/n^c$ fraction of inputs. Then there is a Boolean function family $g \in \mathsf{P}^{\mathsf{NP}_\parallel}$ 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 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 $\oplus^k$.)

Outline of the paper We give some preliminaries in Section 2. We describe the main tools used in our approximate list-decoding algorithm, and give the proof of our Main Theorem in Section 3. The auxiliary lemmas used in the proof of Main Theorem are proved in Section 4. Applications to uniform hardness amplification are given in Section 5. We give concluding remarks in Section 6. The Appendix contains combinatorial bounds on the list size for the truncated Hadamard codes, XOR codes, and direct product codes.

2 Preliminaries

2.1 Notation

For an integer $k > 0$, we will sometimes denote a set $\{1, \ldots, k\}$ by $[k]$. We use $\|v\|_1$ to denote the ℓ_1-norm of a vector $v = (v_1, \ldots, v_n)$ where $\|v\|_1 = \sum_{i=1}^n |v_i|$.

2.2 Some definitions and theorems

Definition 3 (Statistical Distance)**.** Given two distributions D_1 and D_2 over $\{0,1\}^n$, the *statistical distance* between them is defined as half of the ℓ_1-norm of the vector $D_1 - D_2$, i.e.,

$$Dist(D_1, D_2) = \frac{1}{2} \cdot \sum_{x \in \{0,1\}^n} |D_1(x) - D_2(x)|,$$

where $D_i(x)$ denotes the probability of sampling x from distribution D_i, for $i \in \{1, 2\}$. Equivalently,

$$Dist(D_1, D_2) = \max_{T \subseteq \{0,1\}^n} |D_1(T) - D_2(T)|,$$

where $D_i(T)$ denotes the probability mass assigned by D_i to the set T. For $\epsilon > 0$, we say that distributions D_1 and D_2 are *statistically ϵ-close* if $Dist(D_1, D_2) \leqslant \epsilon$.

Definition 4 (Shannon Entropy)**.** The Shannon Entropy of a distribution D over a finite set Λ is defined as

$$H(D) = -\sum_{x \in \Lambda} D(x) \log D(x).$$

Note that the Shannon entropy of the uniform distribution over a set Λ is $\log_2 |\Lambda|$. Usually, we will be interested in distributions over the set of binary strings, i.e., $\Lambda = \{0,1\}^n$. In this case, the Shannon entropy of the uniform distribution over all n-bit strings is exactly n.

We will use the following result from information theory which upperbounds the statistical distance between a distribution P and the uniform distribution by the entropy deficiency of P; see, e.g., [CT91, Lemma 12.6.1].

Lemma 5. *Let P be any probability distribution over a finite set Λ, and let U be the uniform distribution over Λ. Then*

$$\|P - U\|_1^2 \leqslant (2 \ln 2)(H(U) - H(P)),$$

where H is the Shannon entropy.

For a universe S of size N and a subset $T \subseteq S$ of size m, let R be a uniformly random subset of S of size n. The random variable $X = |R \cap T|$ is distributed according to the *hypergeometric distribution* with parameters N, m, and n. The expected value of X is nm/N. The Hoeffding bound [Hoe63] says that this random variable is highly concentrated around its expectation. The proofs of the following versions of the Hoeffding bound can be found, for example, in [JLR00, Theorem 2.10].

Theorem 6 (Hoeffding Bounds for Hypergeometric Distribution)**.** *Let X be a random variable which follows the hypergeometric distribution with parameters N, m, n. Let $\lambda = nm/N$ be the expectation of X. Then we have the following inequalities:*

1. *for $c > 1$ and $x \geqslant c\lambda$,*
$$\mathbf{Pr}\left[X \geqslant x\right] \leqslant e^{-c'x},$$
where $c' = \log c - 1 + 1/c > 0$; in particular, $\mathbf{Pr}[X \geqslant x] \leqslant e^{-x}$ for $x \geqslant 7\lambda$,

2. *for $\alpha \geqslant 0$,*
$$\mathbf{Pr}\left[X \leqslant (1-\alpha)\lambda\right] \leqslant e^{-\alpha^2\lambda/2},$$

3. *for $0 < \alpha \leqslant 3/2$,*
$$\mathbf{Pr}[|X - \lambda| \geqslant \alpha\lambda] \leqslant 2e^{-\alpha^2\lambda/3},$$

4. *for any $\alpha \geqslant 0$,*
$$\mathbf{Pr}[X \geqslant \lambda + \alpha] \leqslant e^{-2\alpha^2/n}.$$

For independent identically distributed random variables $X_1, \ldots, X_n$, where each X_i is 1 with probability p, and 0 with probability $1-p$, their sum $X = \sum_{i=1}^n X_i$ is distributed according to the *binomial distribution* with parameters n and p. The expectation of X is $\lambda = np$. The Chernoff bound [Che52] gives high concentration of X around its expectation λ. In fact, all bounds of Theorem 6 hold also for the case of a binomially distributed random variable X with expectation λ, and we will be using these bounds as the Chernoff bounds in the paper.

For *pairwise independent* identically distributed random variables $X_1, \ldots, X_n$, where each X_i is 1 with probability p, and 0 with probability $1-p$, their sum $X = \sum_{i=1}^{n} X_i$ is also concentrated around the expectation $\lambda = np$. The Chebyshev inequality bounds the deviation as follows: for $x > 0$, we have $\mathbf{Pr}[|X - \lambda| \geqslant x] \leqslant np(1-p)/x^2$.

Finally, we will often use Markov-style averaging arguments. Two of the most common ones we need are as follows. For an event E depending on independent random variables x and y, if $\mathbf{Pr}_{x,y}[E] \geqslant \epsilon$, then with probability at least $\epsilon/2$ over x it is the case that event E holds with probability at least $\epsilon/2$ over y. Also, if $\mathbf{Pr}_{x,y}[E] \geqslant 1-\gamma$, then with probability at least $1-\sqrt{\gamma}$ over x it is the case that event E holds with probability at least $1-\sqrt{\gamma}$ over y.

3 Overview and the proof of Main Theorem

Here we explain the general structure of our proof of Theorem 1. As mentioned in Introduction, we would like to use the learning algorithm $LEARN$ of [IW97]. This algorithm can construct a circuit C' approximately computing an n-variable Boolean function f when given as input a circuit C computing the direct product f^k on a "non-trivial" fraction ϵ of inputs to f^k. This is a *learning* algorithm since it requires the values of the function f on a few randomly independently chosen inputs.

The algorithm from [IW97] works as follows. Given an input z, place z in a random position of a random k-tuple $(x_1, \ldots, x_k)$, where we assume that we know the values of f at $x_1, \ldots, x_k$. Run circuit C on this k-tuple. Depending on the number of correct answers given by C for the x_is, probabilistically form a guess for the value of f at z. Repeat this for $t \approx 1/\epsilon^2$ iterations, and output the value equal to the majority of the guesses.

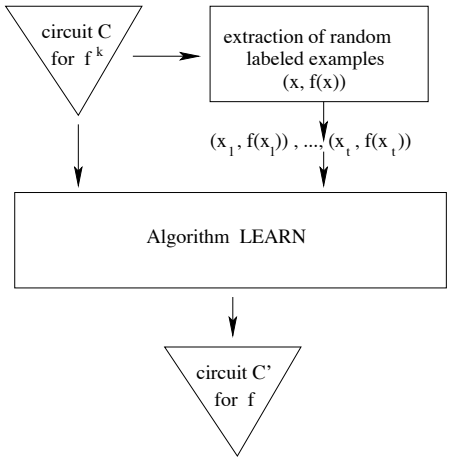
Note that the described algorithm requires $t(k-1)$ pairs $(x, f(x))$, for independent uniformly random xs. Getting such values of f is easy in the nonuniform setting of [IW97]. One just uses a simple counting argument to argue the existence of a small (polynomial in $1/\epsilon$) set of inputs such that algorithm $LEARN$ works almost as well when given the values of f on these fixed inputs. Then one can use nonuniform advice to specify theses inputs and the corresponding values of f.

In the uniform setting, we cannot use as advice the values of f on poly$(1/\epsilon)$ inputs. Instead, *we use the circuit C itself* in order to generate sufficiently many random labeled examples of the form $(x, f(x))$ required by algorithm $LEARN$. Then we can simply run algorithm $LEARN$ on input C and the generated random examples, obtaining the requisite circuit C' that approximately computes f. This approach is summarized in the diagram given in Figure 1 below.

In reality, there are a number of technical issues one needs to handle in order to make the suggested approach work. The main issue is the extraction of random labeled examples $(x, f(x))$ from the circuit C. A natural idea is to run C on a random k-tuple $(x_1, \ldots, x_k)$, where each $x_i \in \{0,1\}^n$, collect the k-tuple of outputs $(b_1, \ldots, b_k)$ of C, and finally output $(x_1, b_1), \ldots, (x_k, b_k)$. Since C is assumed to compute f^k correctly on at least ϵ fraction of input k-tuples, we get with probability ϵ the output of the form $(x_1, f(x_1)), \ldots, (x_k, f(x_k))$.

Note that the given sampling algorithm produces a sequence of *correct* labeled examples only with probability ϵ. However, such a low probability is not a problem for us. If it were the case that with probability poly(ϵ) our sampling algorithm produces enough random labeled examples, then we could conclude that with probability poly(ϵ) the circuit C' produced by algorithm $LEARN$ is good (i.e., it approximates f well). Indeed, the the correctness analysis of $LEARN$ given in [IW97] shows that the circuit C' produced by $LEARN$ is good with probability $1 - o(1)$, when $LEARN$ is given a random sample of sufficiently many labelled examples of the from $(x, f(x))$. By our assumption, we get such a random sample of labelled examples with probability poly(ϵ). So, we

Figure 1. Schematic diagram of the decoding algorithm.



get a good circuit C' with probability $(1 - o(1))\text{poly}(\epsilon) \geqslant \text{poly}(\epsilon)$.

There are two problems with the suggested sampling algorithm. First, the k examples it produces (when successful) are *correlated*. They correspond to those ϵ fraction of k-tuples of inputs where circuit C computes f^k correctly. The uniform distribution over these k-tuples is certainly not uniform over all k-tuples, unless $\epsilon = 1$. On the other hand, algorithm $LEARN$ requires the values of f on $t = \Omega(1/\epsilon^2)$ *independent* random inputs $x_1, \ldots, x_t$.

The second problem is that, even if the samples produced by the suggested sampling algorithm were completely independent, there may not be enough of them. As mentioned above, for algorithm $LEARN$ to work, we need to generate at least $1/\epsilon^2$ labeled examples (actually, a bit more than that, but we'll address this point later). If the given circuit C ϵ-computes f^k for $k \geqslant 1/\epsilon^2$, then the number of examples our sampling algorithm produces is enough. However, if $k < 1/\epsilon^2$, we need to do more work.

Next we explain our solution of these two problems.

3.1 Extracting almost independent random examples

Recall our problem. We are given a circuit C that correctly computes the direct product f^k on at least ϵ fraction of input k-tuples. We want to produce some number t of independent random labeled examples $(x_1, f(x_1)), \ldots, (x_t, f(x_t))$.

Let $G \subseteq \{0,1\}^{nk}$ be the set of those k-tuples where C correctly computes f^k. By our assumption, the weight of G in the universe $\{0,1\}^{nk}$ is at least ϵ. For each $i = 1, \ldots, k$, let G_i be the projection of G to the ith coordinate, i.e., G_i is the set of n-bit strings that can occur in the ith position of a k-tuple from G. Clearly, it cannot be the case that *all* G_is are simultaneously of weight less than $\epsilon^{1/k}$ (in the universe $\{0,1\}^n$), since in that case the weight of G would be less than ϵ.

To develop intuition, let us assume that G is a direct product of $G_1, \ldots, G_k$ and that each G_i is of weight at least $\epsilon^{1/k}$. Now consider the following modified sampling algorithm: Pick a random k-tuple $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^{nk}$; run C on input $\bar{x}$, obtaining the k-tuple of outputs $(b_1, \ldots, b_k)$; pick a random index [1] $j \in \{1, \ldots, k\}$; output the pair (x_j, b_j).

[1] Under our simplifying assumptions on the structure of G, it is not necessary to pick j at random. However, a

Obviously, with probability ϵ the output produced by this new sampling algorithm is a correct example $(x, f(x))$. What is more, conditioned on the random k-tuple falling into the set G, the distribution of x (i.e., the first element of the output pair $(x, f(x))$) is *statistically close to uniform*. The distance from the uniform distribution is at most $1 - \epsilon^{1/k}$, since we assumed that each G_i is of weight at least $\epsilon^{1/k}$ and that G is a direct product of G_is. Observe that as k gets larger, the distribution of x gets closer to the uniform distribution.

Thus, the described sampling procedure allows us to produce (modulo some simplifying assumptions on the structure of G) a single random example $(x, f(x))$ with x being distributed almost uniformly, conditioned on sampling a k-tuple from G.

To get more random examples, one might try running the described sampling procedure multiple times. However, the probability that t independent runs of the sampling procedure produce t good samples will be at most ϵ^t (since a single run succeeds with probability only about ϵ). This is impractical unless t is a constant, but in our case we need a super-constant $t > 1/\epsilon^2$.

A better way to sample more examples is as follows. View an input k-tuple as a $\sqrt{k}$-tuple of $\sqrt{k}$-tuples, i.e., $\bar{x} = (y_1, \ldots, y_{\sqrt{k}})$ where each y_i is in $\{0,1\}^{n\sqrt{k}}$. Run a circuit C on a random $\bar{x}$, and output a random $\sqrt{k}$-subtuple y_j with the corresponding values of C on y_j. The same analysis as above (again under the simplifying assumption on the structure of G) implies that this sampling procedure yields $\sqrt{k}$ examples $(x_1, f(x_1)), \ldots, (x_{\sqrt{k}}, f(x_{\sqrt{k}})$ such that the *entire tuple* $x_1, \ldots, x_{\sqrt{k}}$ is distributed statistically $(1 - \epsilon^{1/\sqrt{k}})$-close to uniform, conditioned on $\bar{x}$ falling in G. So, with probability ϵ, we obtain $\sqrt{k}$ almost independent random labeled examples.

Our discussion above was assuming a nice structure of the set G. In general, G is not so nicely structured, but, nonetheless, the given intuition is correct, and the sampling algorithm just described will still work, albeit with a slightly different bound on the statistical distance from the uniform distribution. We have the following Sampling Lemma, whose proof we postpone till later.

Lemma 7 (Sampling Lemma: Simple Case)**.** *Let C be a circuit that ϵ-computes the direct product f^k, for some Boolean function $f : \{0,1\}^n \to \{0,1\}$. Let $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^{nk}$ 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 at least ϵ such that, conditioned on E, the distribution of $\bar{x'}$ is α-close to the uniform distribution over $\{0,1\}^{n\sqrt{k}}$, where $\alpha \leqslant 0.6\sqrt{\log(1/\epsilon)/\sqrt{k}}$.*

3.2 Increasing the number of extracted random examples

Now we explain how to deal with problem that a given circuit C ϵ-computes the direct product f^k for $k < 1/\epsilon^2$, and so C is not immediately useful to obtain enough random examples required by algorithm $LEARN$. Let $t > 1/\epsilon^2$ be the actual number of random examples used by $LEARN$. Note that if it were the case that $k \geqslant t^2$, then we could use the sampler from Lemma 7 to obtain the requisite t samples.

Suppose that $k < t^2$. In that case, what we would like to do is take the circuit C that approximately computes f^k and construct a new circuit C^{amp} that approximately computes f^{t^2}. This can be viewed as *amplifying* the direct product computation: being able to approximately compute f^k implies being able to approximately compute $f^{k'}$ for $k' > k$.

Unfortunately, we do not know how to achieve such direct product amplification. However, we can get something weaker, which is still sufficient for proving our Main Theorem. The circuit C^{amp}

random choice of j will be useful for the case of an arbitrary G.

we obtain will approximately compute a $(1-\gamma^{amp})$-Direct Product $f^{k'}$, for some small parameter γ^{amp}. That is, on a certain fraction of input k'-tuples, the constructed circuit C^{amp} will be correct in *almost all* positions $1 \leqslant i \leqslant k'$ rather than *all* positions. It turns out that we can also weaken our assumption on the initial circuit C and also just require that C approximately computes $(1-\gamma)$-Direct Product f^k, for some $\gamma \geqslant 0$. We have the following lemma, whose proof we will give later in the paper.

Lemma 8 (Direct Product Amplification)**.** *For every $k \leqslant k' \in \mathbb{N}$, $\gamma > 0$, and $\epsilon > e^{-k^{0.001}}$, there is a randomized algorithm A with the following property. Let f be any n-variable Boolean function such that a circuit C ϵ-computes $(1-\gamma)$-Direct Product f^k. Given C, algorithm A outputs with probability at least ϵ' a circuit C^{amp} that ϵ'-computes $(1-\gamma')$-Direct Product $f^{k'}$, where*

- $\epsilon' = \epsilon^{\text{poly}\log_k k'}$, *and*
- $\gamma' \leqslant (\gamma + k^{-0.4})\text{poly}\log_k k'$.

The running time of A, and hence also the size $|C^{amp}|$ of C^{amp}, is at most $(|C|/\epsilon)^{\text{poly}\log_k k'}$.

Observe that for the case where $\epsilon > \text{poly}(1/k)$, the number of random examples we need to run algorithm $LEARN$ is about $1/\epsilon^2 = \text{poly}(k)$. So we need to amplify direct product for f from k to $k' = \text{poly}(k)$. For such ϵ and k', the Direct Product Amplification theorem above yields that $\epsilon' = \text{poly}(\epsilon)$, the running time of A is at most $\text{poly}(|C|/\epsilon)$, and $\gamma' \leqslant O(\gamma + k^{-0.4})$. Assuming that $\gamma < k^{-0.4}$, we get that $\gamma' < k^{-0.3}$.

The only remaining question is how to use this circuit C^{amp} to extract random examples. Our sampling algorithm from Lemma 7 requires access to a circuit for a direct product of f that is correct on *all* positions for at least ϵ fraction of input tuples, rather than just *almost all* positions. What kind of samples do we get when using such an imperfect direct-product circuit C^{amp}? Are these samples going to be good enough to run algorithm $LEARN$? It turns out that the answer to the last question is yes, once we modify algorithm $LEARN$ appropriately. We shall give more details in the next subsection.

3.3 Algorithm $LEARN$ with faulty examples

It is not difficult to argue (and we will formalize it later) that the same sampling algorithm of Lemma 7 will produce a sequence of samples $(x_1, b_1), \ldots, (x_{\sqrt{k}}, b_{\sqrt{k}})$ such that, conditioned on a certain event E that occurs with probability $\Omega(\epsilon)$, the distribution of all x_is is statistically close to uniform, *and* for almost all $1 \leqslant i \leqslant \sqrt{k}$ it is the case that $b_i = f(x_i)$. That is, almost all of the produced pairs (x_i, b_i) will be correct labeled examples $(x_i, f(x_i))$. Can one use these slightly imperfect examples as input to algorithm $LEARN$? We show that one indeed can, after appropriately modifying the algorithm $LEARN$ to take into account potential inaccuracy in provided random examples.

Before stating the result about our modification of algorithm $LEARN$ from [IW97], we outline how the original algorithm from [IW97] works. Given a circuit C that ϵ-computes f^k, the circuit C' approximately computing f does the following: On input z, randomly sample $t(k-1)$ labeled examples $(x, f(x))$, where t is a parameter of the algorithm. Think of these examples as t blocks of $(k-1)$ pairs $(x, f(x))$ each, so that the ith block is $(x_{i,1}, f(x_{i,1})), \ldots, (x_{i,k-1}, f(x_{i,k-1}))$. For every block $1 \leqslant i \leqslant t$, pick a random position $1 \leqslant j_i \leqslant k$, and form a k-tuple $\bar{x}^i = (x_{i,1}, ..., x_{i,j_i-1}, z, x_{i,j_i}, \ldots, x_{i,k-1})$, with z in the j_ith position. Run circuit C on the tuple $\bar{x}^i$. Depending on the number of correct answers C gives in positions other than j_i, probabilistically assign the variable v_i either the answer of C in position j_i, or a random bit. Finally, output the majority value over all v_i, for $1 \leqslant i \leqslant t$.

The key parameter of this algorithm is the probability with which to believe or not to believe the answer of C in the j_ith position of block i. That probability is inversely exponential in the number of incorrect answers that C gives for the other positions in the block; note that we can verify the correctness of C's answers in these other positions since we know the corresponding values of f.

Suppose we use the same algorithm as above to construct C', but with imperfect samples such that, for each block i, we know the values of f on *almost all* strings $x_{i,1}, \ldots, x_{i,k-1}$ in the block. It is possible to modify the value of the probability with which one believes the answer of C in the j_ith position so that the output circuit C' still approximates the function f well.

Before stating the actual result, we need some definitions. For parameters k, n, t, let D be a probability distribution over t blocks where each block consists of $k-1$ pairs (x, b_x), with $x \in \{0,1\}^n$ and $b_x \in \{0,1\}$; that is, D is a probability distribution over $(\{0,1\}^n \times \{0,1\})^{(k-1)t}$. Think of D as a distribution over t blocks of $(k-1)$ samples required by algorithm $LEARN$. Let D_x be the probability distribution over $\{0,1\}^{n(k-1)t}$ obtained from D by keeping only x from every pair (x, b_x); that is, D_x is the distribution over the inputs to f in all t blocks. For parameters $0 \leqslant \kappa, \gamma \leqslant 1$, we say that the distribution D is (γ, κ)-*good* if

1. for every sample from D, each block $(x_1, b_1), \ldots, (x_{k-1}, b_{k-1})$ of the sample is such that $b_j = f(x_j)$ for at least $(1-\gamma)$ fraction of $j \in [k-1]$, and
2. the distribution D_x is statistically κ-close to the uniform distribution over $\{0,1\}^{n(k-1)t}$.

The next lemma says that an appropriate modification of the algorithm from [IW97] works when given samples from a distribution D that is good according to the definition above.

Lemma 9 (Analysis of Algorithm $LEARN$)**.** *For any $\mu \geqslant \nu > 0$, $\kappa \geqslant 0$, $\epsilon > e^{-k^{\nu/3}}$, $\rho = \frac{\log(1/\epsilon)}{k} + k^{-\nu/2}$, and $t = O((\log 1/\rho)/\epsilon^2)$, there is a probabilistic algorithm $LEARN$ satisfying the following. Let f be an n-variable Boolean function such that a circuit C ϵ-computes $(1-k^{-\mu})$-Direct Product f^k, and let D be a probability distribution over $(\{0,1\}^n \times \{0,1\})^{(k-1)t}$ that is $(k^{-\nu}, \kappa)$-good. Then algorithm $LEARN$, given as input C and a random sample from D, outputs with probability at least $1 - O(\rho) - \kappa$ a circuit C' that computes f on at least $1 - O(\rho)$ fraction of inputs. The running time of $LEARN$, and hence also the size of the circuit C', is at most* $\mathrm{poly}(|C|, 1/\epsilon)$.

Remark 10. The parameters we achieve in Lemma 9 are somewhat weaker than those of the original algorithm of [IW97] where one assumes $\mu = \nu = 1$, $\kappa = 0$, and no restriction on how small ϵ can be. In [IW97], the value ρ is $\frac{\log(1/\epsilon)}{k}$.

3.4 Proof of Main Theorem

Now we can give the proof of our Main Theorem (Theorem 1), using the lemmas above; the proofs of these lemmas will be given in later sections of the paper.

The plan is to use Direct Product Amplification of a given circuit C to obtain a new circuit C^{amp} for a larger direct product of f, extract random labeled examples from C^{amp}, and finally run $LEARN$ on C and extracted random examples. One technicality is that we need to extract examples from a circuit C^{amp} that computes only $(1-\gamma)$-Direct Product of f, for some $\gamma > 0$, whereas our Sampling Lemma (Lemma 7) is for the case where $\gamma = 0$. We will actually need the following version of the Sampling Lemma, which works for the case of nonzero γ.

Lemma 11 (Sampling Lemma: General Case)**.** *Let C be a circuit that ϵ-computes $(1-\gamma)$-direct product f^k, for some Boolean function $f : \{0,1\}^n \to \{0,1\}$. Let $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^{nk}$ 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*, the following two conditions hold:*

1. *the distribution of* $\bar{x'}$ *is statistically* α*-close to the uniform distribution over* $\{0,1\}^{n\sqrt{k}}$*, where* $\alpha \leqslant 0.6\sqrt{\log(1/\epsilon)/\sqrt{k}} + e^{-\Omega(k^{0.1})}$*, and*

2. *for all but at most* $O(\gamma) + k^{-0.4}$ *fraction of elements* x_{i_j}*s in* $\bar{x'}$*, it is the case that* $b_{i_j} = f(x_{i_j})$*.*

Now we can continue with our proof of Theorem 1. Let C be a given circuit that ϵ-computes the $(1-k^{-\mu})$-Direct Product f^k of an n-variable Boolean function f, where $\mu > 0$ and $\epsilon > e^{-k^\alpha}$ for $\alpha = \min\{\mu/1000, 0.0001\}$. Set $\nu = 0.9 * \min\{\mu, 0.4\}$. Let $\rho = \frac{\log(1/\epsilon)}{k} + k^{-\nu/2}$ and $t = O((\log 1/\rho)/\epsilon^2)$; i.e., $t(k-1)$ is the total number of labeled examples required by the modified algorithm $LEARN$ of Lemma 9. Set $k' = (t(k-1))^2$. We do the following.

1. Apply the Direct Product Amplification algorithm of Lemma 8 to the circuit C, with the parameters k, k' as above and $\gamma = k^{-\mu}$. We get with probability $\epsilon' = \epsilon^{\mathrm{poly}\log_k k'}$ a new circuit C^{amp} that ϵ'-computes $(1-\gamma^{amp})$-Direct Product $f^{k'}$, for $\gamma^{amp} \leqslant (k^{-\mu} + k^{-0.4})\mathrm{poly}\log_k k'$. The quantity $\mathrm{poly}\log_k k'$ can be upperbounded by $k^{\min\{\mu/100, 0.001\}}$. So, $\gamma^{amp} \leqslant k^{-0.99\min\{\mu, 0.4\}}$.

2. Apply the Sampling Lemma (Lemma 11) to the circuit C^{amp}, obtaining $\sqrt{k'} = t(k-1)$ samples of the form $(x, b_x) \in \{0,1\}^n \times \{0,1\}$. With probability $\Omega(\epsilon')$ an event E occurs such that, conditioned on E, the distribution on xs is statistically $0.6\sqrt{(\log 1/\epsilon')/\sqrt{k'}} = o(1)$-close to uniform, and all but at most $\gamma' = O(\gamma^{amp}) + k^{-0.4}$ fraction of samples are correct labeled examples of the form $(x, f(x))$.

3. Randomly partition the $t(k-1)$ samples into t blocks. This will ensure that each block has at most $2\gamma'$ incorrect examples, with high probability. Indeed, for a fixed block, the probability that it gets more than twice the expected number $\gamma'(k-1)$ of incorrect examples is, by the Hoeffding bound of Theorem 6, at most $e^{-\Omega(\gamma'(k-1))}$. The latter is at most $e^{-\Omega(k^{0.6})}$ since $\gamma' \geqslant k^{-0.4}$. By the union bound, the probability that at least one of the t blocks gets more than twice the expected number of incorrect examples is at most $te^{-\Omega(k^{0.6})}$.

   For $t = O((\log 1/\rho)/\epsilon^2)$, $\rho = (\log 1/\epsilon)/k + k^{-\nu/2}$, and $\epsilon \geqslant e^{-k^{0.001}}$, we get $\rho \leqslant k^{-\Omega(1)}$ and $t \leqslant O(\log k/\epsilon^2)$. Hence, the probability that any block gets more than $2\gamma' \leqslant k^{-\nu}$ fraction of incorrect examples is at most $O(\log k/\epsilon^2) * e^{-\Omega(k^{0.6})}$, which is exponentially small in $k^{\Omega(1)}$.

4. Suppose that event E occurred. Then the last two steps ensure that we have with probability $1 - e^{-k^{\Omega(1)}}$ a sample from a $(k^{-\nu}, o(1))$-good distribution over t blocks of $(k-1)$ pairs (x, b_x). Applying algorithm $LEARN$ from Lemma 9 to the circuit C and this sample, we get with probability $1 - O(\rho) - o(1) = 1 - o(1)$ a circuit C' that $(1 - O(\rho))$-computes f. Lifting the conditioning on E, we conclude that such a circuit C' is produced with probability at least $\Omega(\epsilon')$.

The running time of the algorithm described in the four steps above is at most $(|C|/\epsilon)^{\mathrm{poly}\log_k k'}$, which can be upperbounded by $(|C|/\epsilon)^{\mathrm{poly}\log_k 1/\epsilon}$. The probability $\Omega(\epsilon')$ of getting a good circuit C' can be lowerbounded by $\epsilon^{\mathrm{poly}\log_k 1/\epsilon}$.

This finishes the proof of Theorem 1, modulo the proofs of Lemmas 8, 9, and 11, which will be given in the following sections of the paper.

4 Our tools

4.1 Proof of the Sampling Lemma

Here we give the proof of Lemma 11. For the ease of reference, we re-state this lemma below.

Lemma 11 (Sampling Lemma: General Case). *Let C be a circuit that ϵ-computes $(1-\gamma)$-direct product f^k, for some Boolean function $f : \{0,1\}^n \to \{0,1\}$. Let $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^{nk}$ 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, the following two conditions hold:*

1. *the distribution of $\bar{x'}$ is statistically α-close to the uniform distribution over $\{0,1\}^{n\sqrt{k}}$, where $\alpha \leqslant 0.6\sqrt{\log(1/\epsilon)/\sqrt{k}} + e^{-\Omega(k^{0.1})}$, and*
2. *for all but at most $O(\gamma) + k^{-0.4}$ fraction of elements x_{i_j}s in $\bar{x'}$, it is the case that $b_{i_j} = f(x_{i_j})$.*

We need the following result implicit in [Raz98].

Lemma 12. *Let Λ be any finite set. Let $G \subseteq \Lambda^m$ be any subset of m-tuples of elements of Λ such that G has density ϵ in the set Λ^m. Let U be the uniform distribution on elements of Λ, and let D be the distribution defined as follows: pick a tuple $(x_1, \ldots, x_m)$ uniformly from the set G, pick an index i uniformly from $[m]$, and output x_i. Then $Dist(U, D) \leqslant 0.6\sqrt{\frac{\log 1/\epsilon}{m}}$.*

Proof. Let $\bar{X} = (X_1, \ldots, X_m)$ 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 $X_1, \ldots, X_m$, i.e., $\mathbf{Pr}[X = x] = \frac{1}{m}\sum_{i=1}^{m} \mathbf{Pr}[X_i = x]$. By the concavity of the entropy function, we obtain

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

Since the sum of entropies is lowerbounded by the joint entropy, we get

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

Since $\bar{X} = (X_1, \ldots, X_m)$ is uniformly distributed over G, its Shannon entropy is exactly $\log_2 |G| = m \log_2 |\Lambda| - \log_2(1/\epsilon)$. Combining this with Eq. (1) above, we get

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

Finally, we use Lemma 5 to conclude that

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

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

Now we can prove Lemma 11.

Proof of Lemma 11. Let G be the set of those ϵ-fraction of k-tuples $\bar{x}$ where C computes f^k for all but γ fraction of positions. A random k-tuple $\bar{x}$ falls into the set G with probability at least ϵ. Call this event E_1. Conditioned on E_1, this tuple is uniformly distributed over G and hence, by Lemma 12, we conclude that a random $\sqrt{k}$-subtuple of $\bar{x}$ is distributed almost uniformly, with the statistical distance from the uniform distribution less than $0.6\sqrt{\log(1/\epsilon)/\sqrt{k}}$.

Observe that every tuple in G has at most γ fraction of "bad" elements where C disagrees with the function f. If we pick a random $\sqrt{k}$-size subtuple, the probability that it contains more than $7\gamma + k^{-0.4}$ fraction of bad elements will be at most $e^{-(7\gamma+k^{-0.4})\sqrt{k}} \leqslant e^{-k^{0.1}}$ by the Hoeffding bound of item (1) of Theorem 6.

Let E_2 denote the event that a random subtuple contains at most $7\gamma + k^{-0.4}$ fraction of "bad" elements. We get that conditioned on E_1, the probability of E_2 is at least $1 - e^{-k^{0.1}}$. Let the event E be the conjunction of E_1 and E_2. Clearly, the probability of E is at least $\epsilon(1 - e^{-k^{0.1}}) = \Omega(\epsilon)$. It is also clear that conditioned on E, the output subtuple satisfies item (2) of the lemma. Finally, it is easy to argue that since E_2 has high probability conditioned on E_1, the distribution of random elements of $\bar{x}$ conditioned on E has statistical distance from the uniform distribution at most $0.6\sqrt{\log(1/\epsilon)/\sqrt{k}} + e^{-\Omega(k^{0.1})}$. Hence, item (1) of the lemma also holds. □

For the analysis of our decoding algorithm, we shall also need the following variant of Lemma 12 for the case of *sets* rather than tuples, i.e., for the case of sampling *without* replacement.

Lemma 13. *Let Λ be any finite set, and let S be the set of all m-size subsets of Λ. Let $G \subseteq S$ be any subset that has density at least ϵ in the set S. Let U be the uniform distribution over Λ, and let D be the distribution defined as follows: pick a set $s \in G$ uniformly at random and output a random element of s. Then $Dist(U, D) \leqslant O\left(\sqrt{\frac{\log(m/\epsilon)}{m}}\right)$.*

The proof idea is as follows. Assume that there is a subset $T \subseteq \Lambda$ such that the probability of T under D is bigger than that under U by some δ; the case where it is smaller by δ is similar. We will show that in this case a random set from G is expected to contain at least $\mu = m|T|/|\Lambda| + m\delta$ elements from the set T. Also, the probability that a random set from G contains at least its expected number of elements from T is at least about $1/m$. Since G has density at least ϵ in S, we conclude that a random set from S has at least μ elements from T with probability at least ϵ/m. But, a random set from S follows the hypergeometric distribution and is expected to contain only $m|T|/|\Lambda|$ elements from T. The probability that the number of elements from T exceeds the expectation by $m\delta$ can be upperbounded by $e^{-\Omega(m\delta^2)}$, using the Hoeffding inequality. Hence, $\epsilon/m \leqslant e^{-\Omega(m\delta^2)}$, whence the required upper bound on δ follows.

We now give the details of the proof. First, we show the following simple property of integer-valued random variables.

Claim 14. *Let X be any integer-valued random variable taking values in the set $\{0, 1, \ldots, m\}$. Let $\mu = \boldsymbol{E}[X]$ be the expectation of X. Then $\mathbf{Pr}[X \geqslant \lfloor\mu\rfloor] \geqslant 1/m$ and $\mathbf{Pr}[X \leqslant \lceil\mu\rceil] \geqslant 1/(m+1)$.*

Proof. By definition, $\mu = \sum_{i=0}^{m} i \cdot \mathbf{Pr}[X = i]$. We can split this sum into two sums as follows:

$$\mu = \sum_{0<i<\lfloor\mu\rfloor} i \cdot \mathbf{Pr}[X = i] + \sum_{\lfloor\mu\rfloor \leqslant i \leqslant m} i \cdot \mathbf{Pr}[X = i].$$

Using the fact that X is integer-valued, we can bound these two sums by

$$(\mu-1)\cdot\sum_{0<i<\lfloor\mu\rfloor}\mathbf{Pr}[X=i]+m\cdot\sum_{\lfloor\mu\rfloor\leqslant i\leqslant m}\mathbf{Pr}[X=i]\leqslant(\mu-1)+m\cdot\mathbf{Pr}[X\geqslant\lfloor\mu\rfloor].$$

This implies that $\mathbf{Pr}[X\geqslant\lfloor\mu\rfloor]\geqslant 1/m$, as claimed in the first inequality of the lemma.

To prove the second inequality, we observe that $\mathbf{Pr}[X>\lceil\mu\rceil]=\mathbf{Pr}[X\geqslant\lceil\mu\rceil+1]\leqslant\frac{\mu}{\lceil\mu\rceil+1}$, where we used the fact that X is integer-valued and then applied Markov's inequality. Finally, we have $\mathbf{Pr}[X\leqslant\lceil\mu\rceil]\geqslant 1-\frac{\mu}{\lceil\mu\rceil+1}=\frac{\lceil\mu\rceil+1-\mu}{\lceil\mu\rceil+1}\geqslant\frac{1}{m+1}$. □

Now we prove Lemma 13, following the proof outline given above.

Proof of Lemma 13. Suppose that the statistical distance between U and D is δ. It means that there is a statistical test $T\subseteq\Lambda$ such that the probabilities assigned to T by U and D differ by δ. We shall assume that $\mathbf{Pr}_D[T]\geqslant\mathbf{Pr}_U[T]+\delta$; the case where $\mathbf{Pr}_U[T]\geqslant\mathbf{Pr}_D[T]+\delta$ is proved similarly.

We will view each set $x=\{x_1,\ldots,x_m\}$ of size m as an increasing sequence of m elements $x_1<\cdots<x_m$, under some fixed ordering of the elements of Λ (say, the lexicographical order). Uniformly sampling a set from G will be denoted by $(x_1,\ldots,x_m)\leftarrow G$. Also, for an event E, we use $\chi[E]$ as the 0-1 indicator variable of E. We have

$$\begin{aligned}\mu\stackrel{def}{=}\mathbf{E}_{\bar{x}=(x_1,\ldots,x_m)\leftarrow G}\left[\sum_{i=1}^{m}[\chi[x_i\in T]\right] &= \sum_{i=1}^{m}\mathbf{E}_{\bar{x}\leftarrow G}[\chi[x_i\in T]]\\ &= m\cdot\mathbf{E}_{i\leftarrow[m],\bar{x}\leftarrow G}[\chi[x_i\in T]]\\ &= m\cdot\mathbf{Pr}_D[T]\\ &\geqslant m\cdot(\mathbf{Pr}_U[T]+\delta)\\ &= m\cdot(|T|/|\Lambda|+\delta).\end{aligned}$$

On the other hand, by Claim 14, we get $\mathbf{Pr}_{\bar{x}\leftarrow G}\left[\sum_{i=1}^{m}\chi[x_i\in T]\geqslant\mu-1\right]\geqslant 1/m$. Since G has weight at least ϵ in the collection S of all size-m subsets of Λ, we have

$$\mathbf{Pr}_{\bar{x}\leftarrow S}\left[\sum_{i=1}^{m}\chi[x_i\in T]\geqslant\mu-1\right]\geqslant\epsilon/m. \tag{2}$$

The expectation of $\sum_{i=1}^{m}\chi[x_i\in T]$ under the uniform distribution of $\bar{x}$ in S is $m|T|/|\Lambda|$, with $\bar{x}$ following the hypergeometric distribution with parameters $|\Lambda|$, $|T|$, and m. Applying the Hoeffding bound of item (4) of Theorem 6 to the left-hand side of Eq. (2), we get

$$\mathbf{Pr}_{\bar{x}\leftarrow S}\left[\sum_{i=1}^{m}\chi[x_i\in T]\geqslant m|T|/|\Lambda|+m\delta-1\right]\leqslant e^{-2(m\delta-1)^2/m}=e^{-2m(\delta-1/m)^2}. \tag{3}$$

Note that if $\delta<2/m$, then the claim in the lemma follows. If $\delta\geqslant 2/m$, then we can upperbound the right-hand side of Eq. (3) by $e^{-m\delta^2/2}$. Combining this with Eq. (2), we conclude that $m\delta^2/2\leqslant\ln(m/\epsilon)$, and so $\delta\leqslant O\left(\sqrt{\frac{\log(m/\epsilon)}{m}}\right)$, as required. □

Remark 15. Our proof of Lemma 13 can be also used for the case of sampling with replacement, yielding another proof of Lemma 12. However, the information-theoretic proof of Lemma 12 given above achieves better parameters for that case.

4.2 Proof of the Direct Product Amplification

Here we will prove Lemma 8. For convenience, we re-state it below.

Lemma 8 (Direct Product Amplification). *For every $k \leqslant k' \in \mathbb{N}$, $\gamma > 0$, and $\epsilon > e^{-k^{0.001}}$, there is a randomized algorithm A with the following property. Let f be any n-variable Boolean function such that a circuit C ϵ-computes $(1-\gamma)$-Direct Product f^k. Given C, algorithm A outputs with probability at least ϵ' a circuit C^{amp} that ϵ'-computes $(1-\gamma')$-Direct Product $f^{k'}$, where*

- $\epsilon' = \epsilon^{\mathrm{poly}\log_k k'}$, *and*
- $\gamma' \leqslant (\gamma + k^{-0.4})\mathrm{poly}\log_k k'$.

The running time of A, and hence also the size $|C^{amp}|$ of C^{amp}, is at most $(|C|/\epsilon)^{\mathrm{poly}\log_k k'}$.

Our proof of Lemma 8 will be recursive. We first show how to get from a circuit approximately computing the direct product f^k a new circuit that approximately computes $f^{k^{1.5}}$. Then we iterate this one-step amplification $O(\log\log_k k')$ times, obtaining a circuit for $f^{k'}$.

We state the lemma about one-step amplification next.

Lemma 16 (One-step Direct Product Amplification)**.** *There is a randomized algorithm A with the following property. Let f be any n-variable Boolean function such that a circuit C ϵ-computes $(1-\gamma)$-Direct Product f^k, where $\epsilon > e^{-k^{0.001}}$. Given C, algorithm A 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 $1/\epsilon$.*

Applying Lemma 16 for $\log_{1.5}\log_k k'$ times, one easily obtains Lemma 8. The rest of this section is devoted to the proof of Lemma 16.

We need the following definition. For a k'-tuple $\bar{x}$ and a subset $S \subseteq \{1,\ldots,k'\}$, we denote by $\bar{x}|_S$ the restriction of $\bar{x}$ to the positions in S; so $\bar{x}|_S$ is a tuple of size $|S|$.

We now give an informal description of our algorithm in Lemma 16. Given an input k'-tuple $\bar{x}$, we randomly choose a k-size set S, run the circuit C on $\bar{x}|_S$, and use the obtained values as if they were equal to $f^k(\bar{x}|_S)$. This roughly gives us the values of f on a portion of $\bar{x}$. To obtain the values of f in the positions of $\bar{x}$ outside the set S, we repeatedly do the following. Pick a random k-size set T and run C on $\bar{x}|_T$. If the values $C(\bar{x}|_S)$ and $C(\bar{x}|_T)$ are mostly consistent on inputs $\bar{x}|_{S\cap T}$, then we trust that the values $C(\bar{x}|_T)$ are mostly correct, and take one such value for a random position in T. This gives us a vote for the value of f on another input in $\bar{x}$. Repeating this process enough times, we collect many votes for many positions in $\bar{x}$. For each position of $\bar{x}$ where we have enough votes, we take the majority value as our guess for the value f in that position. For those (few) positions where we don't have enough votes, we assign the value of f arbitrarily, say by flipping a coin.

The formal description of the algorithm is given in Figure 1.

Now we analyze the algorithm given above. Let $G \subseteq \{0,1\}^{nk}$ be the set of those k-tuples where the circuit C correctly computes f^k in all but at most γ fraction of positions $1 \leqslant j \leqslant k$. In the analysis, we will focus on certain $k' = k^{1.5}$-tuples $\bar{x}$ that have the property that a significant fraction of their subtuples of size k fall in the set G.

We need the following definition. For a given k'-tuple $\bar{x}$, we say that a size-k set $S \subseteq \{1,\ldots,k'\}$ is *α-good* if $C(\bar{x}|_S)$ disagrees with $f^k(\bar{x}|_S)$ in at most α fraction of positions. The following claim says that there are many k'-tuples that have many γ-good sets.

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 ϵ-computes $(1-\gamma)$-Direct Product f^k
PARAMETERS: $\epsilon > e^{-k^{0.001}}$, $\rho = 16\gamma + 3k^{-0.4}$, $timeout = (192k' \ln k')/\epsilon$, $\alpha = 29\gamma + 6k^{-0.4}$.

1. For every $i \in [k']$, set $Answers_i =$*empty string*.
2. Choose $S \subseteq [k']$ of size $|S| = k$ uniformly at random.
3. Let $S = \{j_1, \ldots, j_k\}$ where $1 \leqslant j_1 < j_2 < \cdots < j_k \leqslant k'$.
4. Run $C(\bar{x}|_S) = (b_{j_1} \ldots b_{j_k})$, where $\bar{x}|_S$ denotes the restriction of $\bar{x}$ to the coordinates from S.
5. Repeat lines 6–11 for at most *timeout* times:
6. Choose $T \subseteq [k']$ of size $|T| = k$ uniformly at random.
7. Let $T = \{j'_1, \ldots, j'_k\}$ where $1 \leqslant j'_1 < j'_2 < \cdots < j'_k \leqslant k'$.
8. Compute $C(\bar{x}|_T) = (t_{j'_1} \ldots t_{j'_k})$.
9. Let $m = |\{j \in S \cap T \mid b_j \neq t_j\}|/|S \cap T|$.
10. If $m < \rho$, % *if T is consistent with S*
11. then choose a random $i \in T$ and extend the string $Answers_i$ with the bit t_i.
12. For every $i \in [k']$, let $count_i = |Answers_i|$.
13. Let $total = \sum_{i \in [k']} count_i$.
14. If $count_i < \frac{total}{2k'}$, then set $output_i = r_i$ for a random bit r_i;
15. else set $output_i = \text{Majority} Answers_i$.

Algorithm 1: Direct Product amplifier

Claim 17. *There are at least $\epsilon/2$ fraction of k'-tuples $\bar{x}$ such that*

$$\mathbf{Pr}_{S \subset \{1,\ldots,k'\}:|S|=k}[S \text{ is } \gamma\text{-good}] \geqslant \epsilon/2, \tag{4}$$

where the probability is over uniformly random size-k subsets S of the set $\{1, \ldots, k'\}$.

Proof. Note that for a random k'-tuple $\bar{x}$ and a random size-k subset S, the tuple $\bar{x}|_S$ is uniformly distributed in $\{0,1\}^{nk}$, and so it falls into G with probability at least ϵ. A simple averaging argument completes the proof. □

For the rest of the proof, we fix one particular k'-tuple $\bar{x}$ satisfying Eq. (4) of Claim 17. If we could somehow tell whether a given subset T is γ-good or not, we could just sample enough γ-good subsets T, taking the answer of the circuit C on $\bar{x}|_T$ for a random element of T. Since there are at least $\epsilon/2$ fraction of γ-good subsets for our fixed input $\bar{x}$, this sampling procedure will produce enough samples in time polynomial in $1/\epsilon$. Also, given the density of γ-good subsets, our sampling lemma implies that a random element of a random γ-good subset is almost uniformly distributed among the elements of $\bar{x}$, and hence, our sampling procedure will cover most of the elements of $\bar{x}$ after a small number of iterations.

In reality, we cannot test if a subset T is γ-good. However, we have an "approximate test" that accepts almost all γ-good sets, and such that almost every set it accepts is γ'-good, for some γ' slightly larger than γ. This test is the *consistency test* between a given set T and some fixed set S.

We now define our notion of consistency. For two size-k subsets $S = \{j_1, \ldots, j_k\}$ and $T = \{j'_1, \ldots, j'_k\}$ of $[k']$, let $C(\bar{x}|_S) = (b_{j_1} \ldots b_{j_k})$ and let $C(\bar{x}|_T) = (t_{j'_1} \ldots t_{j'_k})$. We say that a set T is *consistent with* S if $|\{j \in S \cap T \mid b_j \neq t_j\}|/|S \cap T| < 16\gamma + 3k^{-0.4}$.

We will argue that for most γ-good sets S,

1. **[Completeness]** almost every γ-good set T is consistent with S, and

2. **[Soundness]** almost every set T consistent with S is $O(\gamma + k^{-0.4})$-good.

Claim 18 (Completeness)**.** *For at least $1 - e^{-\Omega(k^{0.1})}$ fraction of γ-good sets S,*

$$\mathbf{Pr}_{\gamma\text{-good } T}[T \text{ is consistent with } S] \geqslant 1 - e^{-\Omega(k^{0.1})}.$$

Proof. First we show that

$$\mathbf{Pr}_{\gamma\text{-good } S,\ \gamma\text{-good } T}[T \text{ is consistent with } S] \geqslant 1 - e^{-\Omega(k^{0.1})}. \tag{5}$$

The conclusion of the claim will then follow from Eq. (5) by a straightforward averaging argument.

For S, let us denote by S_B the subset of all those elements $j \in S$ where $C(\bar{x}|_S)_j$ is wrong. Define $S_G = S \setminus S_B$. Similarly, let T_B denote the subset of T where the circuit C makes mistakes, and let $T_G = T \setminus T_B$. By our assumption that S and T are γ-good, we have that both S_B and T_B have weight at most γ in their respective sets. Note that the errors in $S \cap T$ can only come from $S_G \cap T_B$ and $T_G \cap S_B$. We'll upperbound the sizes of these sets by upperbounding the sizes of bigger sets $S \cap T_B$ and $T \cap S_B$, respectively.

Fix a γ-good set T. Note that T_B has density at most $\gamma k/k' = \gamma/\sqrt{k}$ in the universe $\{1, \ldots, k'\}$. A random (not necessarily γ-good) size-k subset S of $\{1, \ldots, k'\}$ is expected to intersect T_B in at most $k\gamma/\sqrt{k} = \gamma\sqrt{k}$ places. By the Hoeffding bound of Theorem 6, we get that

$$\mathbf{Pr}_S[|S \cap T_B| > (7\gamma + k^{-0.4})\sqrt{k}] \leqslant e^{-k^{0.1}}.$$

Conditioned on S being a γ-good set, this probability will be at most factor $2/\epsilon$ larger (since γ-good sets have weight at least $\epsilon/2$ among the k-size subsets of $[k']$). Since $\epsilon > e^{-k^{0.001}}$, the resulting conditional probability is still at most $e^{-\Omega(k^{0.1})}$.

A symmetric argument shows that for all but an exponentially small fraction of γ-good sets T, the intersection $T \cap S_B$ will be at most $(7\gamma + k^{-0.4})\sqrt{k}$ for any fixed γ-good set S. Thus, overall, for all but an exponentially small fraction of γ-good sets S and T, the set $(S \cap T_B) \cup (T \cap S_B)$ will be of size at most $2(7\gamma + k^{-0.4})\sqrt{k}$.

On the other hand, applying the Hoeffding bound of Theorem 6 to the size of the set $S \cap T$, we get that that for all but an exponentially small fraction of γ-good S and T, $(1-0.1)\sqrt{k} \leqslant |T \cap S| \leqslant (1+0.1)\sqrt{k}$. Thus, with probability at least $1 - e^{-\Omega(k^{0.1})}$ over γ-good sets S and T, the fraction of indices in $S \cap T$ where the circuit C gives inconsistent answers will be less than $16\gamma + 3k^{-0.4}$, and so T is consistent with S. □

Claim 19 (Soundness)**.** *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 consistent with S are $\alpha = (29\gamma + 6k^{-0.4})$-good.*

Proof. As in the proof of Claim 18 above, we'll denote by S_B and S_G the subsets of indices of S where the circuit C makes mistakes and is correct, respectively. Similarly we denote by T_B and T_G the corresponding subsets of T. We will show that

$$\mathbf{Pr}_{T,\gamma\text{-good } S}[|T_B| > \alpha k \mid T \text{ is consistent with } S] \leqslant e^{-\Omega(k^{0.1})}. \tag{6}$$

The claim will then follow by a simple averaging argument.

The conditional probability in Eq. (6) can be equivalently written as

$$\frac{\mathbf{Pr}_{T,\gamma\text{-good } S}[|T_B| > \alpha k \text{ and } T \text{ is consistent with } S]}{\mathbf{Pr}_{T,\gamma\text{-good } S}[T \text{ is consistent with } S]}. \tag{7}$$

Since by our assumption γ-good sets T have weight at least $\epsilon/2$ in $[k']$, we can lowerbound the probability in the denominator of (7) by

$$(\epsilon/2)\mathbf{Pr}_{\gamma\text{-good } T,\gamma\text{-good } S}[T \text{ is consistent with } S].$$

By Eq. (5), this is at least $(\epsilon/2)(1 - e^{-\Omega(k^{0.1})}) \geqslant \epsilon/3$. It remains to upperbound the probability in the numerator of (7).

Recall that T is consistent with S if both $|T_B \cap S_G|$ and $|T_G \cap S_B|$ are small relative to $|S \cap T|$. We will consider only $T_B \cap S_G$, and show that with high probability either $|T_B| \leqslant \alpha k$ or $T_B \cap S_G$ is large relative to $|S \cap T|$. Since the latter implies that T is not consistent with S, we will be done with upperbounding the numerator of (7).

First we bound $|S \cap T|$. For every fixed γ-good set S, the Hoeffding bound of Theorem 6 yields that $\mathbf{Pr}_T[|T \cap S| \geqslant 1.1\sqrt{k}] \leqslant e^{-\Omega(\sqrt{k})}$. Clearly the same is also true for a random γ-good set S, i.e.,

$$\mathbf{Pr}_{T,\gamma\text{-good S}}[|T \cap S| \geqslant 1.1\sqrt{k}] \leqslant e^{-\Omega(\sqrt{k})}. \tag{8}$$

Next we observe that $|T_B \cap S_G| = |T_B \cap S| - |T_B \cap S_B| \geqslant |T_B \cap S| - |T \cap S_B|$. Using the Hoeffding bound, we will upperbound $|T \cap S_B|$ and lowerbound $|T_B \cap S|$. For every fixed γ-good set S, the Hoeffding bound of Theorem 6 yields that $\mathbf{Pr}_T[|T \cap S_B| > (7\gamma + k^{-0.4})\sqrt{k}] \leqslant e^{-k^{0.1}}$. The same also holds for a random γ-good S, i.e.,

$$\mathbf{Pr}_{T,\gamma\text{-good } S}[|T \cap S_B| > (7\gamma + k^{-0.4})\sqrt{k}] \leqslant e^{-k^{0.1}}. \tag{9}$$

For every fixed set T_B such that $|T_B| > \alpha k$, a random size-k set S is expected to intersect T_B in more than $\frac{\alpha k}{k'}k = \alpha\sqrt{k}$ places. By the Hoeffding bound of Theorem 6,

$$\mathbf{Pr}_S[|S \cap T_B| \leqslant 0.9\alpha\sqrt{k} \mid |T_B| > \alpha k] \leqslant e^{-\Omega(\alpha\sqrt{k})} \leqslant e^{-\Omega(k^{0.1})},$$

where the latter inequality is because $\alpha \geqslant k^{-0.4}$. Since γ-good sets have weight at least $\epsilon/2$ in the universe of all size-k sets and since $\epsilon > e^{-k^{0.001}}$, we get

$$\mathbf{Pr}_{\gamma\text{-good } S}[|S \cap T_B| \leqslant 0.9\alpha\sqrt{k}] \leqslant (2/\epsilon)e^{-\Omega(k^{0.1})} \leqslant e^{-\Omega(k^{0.1})}. \tag{10}$$

It is easy to see that inequality (10) implies

$$\mathbf{Pr}_{T,\gamma\text{-good } S}[|S \cap T_B| < 0.9\alpha\sqrt{k} \text{ and } |T_B| > \alpha k] \leqslant e^{-\Omega(k^{0.1})}. \tag{11}$$

Combining Eqs. (8), (9), and (11), we get that with probability at least $1 - e^{-\Omega(k^{0.1})}$,

1. $|T_B| \leqslant \alpha k$, or
2. $|S \cap T| < 1.1\sqrt{k}$, $|T \cap S_B| \leqslant (7\gamma + k^{-0.4})\sqrt{k}$, and $|T_B \cap S| \geqslant 0.9\alpha\sqrt{k}$.

In the second case, we have $(|T_B \cap S| - |T \cap S_B|)/|S \cap T| \geqslant (0.9\alpha - 7\gamma - k^{-0.4})\sqrt{k}/(1.1\sqrt{k}) \geqslant 0.8\alpha - 7\gamma - k^{-0.4}$, which is greater than $16\gamma + 3k^{-0.4}$ for our choice of α. Hence, in this case, T is not consistent with S. It follows that the numerator of (7) is at most $e^{-\Omega(k^{0.1})}$. Since we also lowerbounded the denominator of (7) by $\epsilon/3$ and since $\epsilon > e^{-k^{0.001}}$, we obtain inequality (6), as required. □

Using Claims 18 and 19, we immediately get the following.

Claim 20. *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 consistent with* S*, and*
3. *all but* $e^{-\Omega(k^{0.1})}$ *fraction of sets* T *consistent with* S *are* α*-good, for* $\alpha = 29\gamma + 6k^{-0.4}$*.*

We will continue the analysis of our algorithm, assuming that such a set S satisfying properties (1)–(3) of Claim 20 was chosen. First we argue that the algorithm will reach line 11 quite often.

Claim 21. *With probability at least* $1 - o(1)$*, the final value of the variable total of our algorithm is at least* $timeout * \epsilon/6 = 32k' \ln k'$*.*

Proof. By item (2) of Claim 20 and by the fact that there are at least $\epsilon/2$ fraction of γ-good sets T, we know that the probability of reaching line 11 in a single iteration is at least $\epsilon/2(1 - e^{-\Omega(k^{0.1})}) \geqslant \epsilon/3$. Thus, the expected value of *total*, which is the total number of times that line 11 was executed, is at least $timeout * (\epsilon/3)$. By Chernoff, the probability that $total < timeout * \epsilon/6$ is at most $e^{-timeout*\epsilon/24} = e^{-8k' \ln k'}$. So, with high probability, the value *total* is at least $timeout * \epsilon/6$. □

Now we bound the number of errors made by the algorithm in line 15.

Claim 22. *With probability at least* $1/2$*, the number of wrong outputs* $output_i$ *made in line 15 of the algorithm is at most* $9\alpha k'$*.*

Proof. By item (3) of Claim 20, we know that conditioned on extending one of the strings $Answers_i$ in line 11, the probability of extending that string with a wrong bit is at most $\alpha + e^{-\Omega(k^{0.1})} \leqslant 1.1\alpha$. Thus, the expected fraction of wrong bits in the entire collection of the strings $Answers_i$ will be at most 1.1α. Let I denote the set of those $i \in [k']$ where $|Answers_i| > total/(2k')$ and Majority$Answers_i$ is wrong. Then the total fraction of wrong answers in all strings $Answers_i$ is at least $|I|/(4k')$. By the above, this is expected to be at most 1.1α. So, the expected number of those $i \in [k']$ where our algorithm makes a mistake in line 15 is at most $4.4\alpha k'$. Applying the Markov inequality, we get the required claim. □

Next we argue that the number of $i \in [k']$ such that $output_i$ is set randomly in line 14 of the algorithm will be small. The idea is that when we extend the string $Answers_i$ in line 11 of the algorithm, we do it for an index $i \in [k']$ that is distributed almost uniformly over $[k']$. Since line 11 is reached many times, we can conclude that the fraction of indices $i \in [k']$ that were selected too few times will be small.

To argue that an index i chosen in line 11 of the algorithm is distributed almost uniformly, we use Lemma 13.

Claim 23. *With probability at least* $1 - o(1)$*, the fraction of* $i \in [k']$ *such that* $output_i$ *will be randomly set in line 14 is at most* $O(k^{-0.4})$*.*

Proof. Given a sequence of *total* guesses made by the algorithm, we will argue that most $i \in [k']$ have $|Answers_i| \geqslant total/(2k')$.

Recall that the collection of random sets T's consistent with S has weight at least $\epsilon/3$ according to the uniform distribution over all size-k random sets in $[k']$. By Lemma 13, we get that the distribution D over $[k']$ induced by taking a random set T consistent with S and then outputting a

random element of T has statistical distance at most $\beta \stackrel{\text{def}}{=} O(\sqrt{\frac{\log(k/\epsilon)}{k}}) \leqslant O(k^{-0.4})$ to the uniform distribution over $[k']$ (since $\epsilon > e^{-k^{0.001}}$).

By a Markov-style argument, the fraction of those $i \in [k']$ that have probability less that $3/(4k')$ under D is at most 4β. Let J denote the set of the remaining i's. For each $i \in J$, we expect the length of the string $Answers_i$ to be at least $\frac{3}{4k'}total$. By the Chernoff bound, the probability that $|Answers_i| < total/(2k')$ is at most $e^{-total/(16k')}$. By the union bound, the probability that there is at least one $i \in J$ with $|Answers_i| < total/(2k')$ is at most $k'e^{-total/(16k')}$.

By Claim 21, the value $total \geqslant 32k' \ln k'$ with probability $1 - o(1)$. Conditioned on *total* being that large, the probability of having at least one $i \in J$ set to a random bit in line 14 of the algorithm is at most $1/k'$. Thus, with probability $1 - o(1)$, only a set of size at most 4β of indices will be randomly set in line 14. The claim follows. □

Now we can finish the analysis.

Proof of Lemma 16. With probability at least $\epsilon/2$, a random input k'-tuple $\bar{x}$ satisfies Eq. (4) of Claim 17. By Claim 20, with probability at least $\epsilon/3$ the set S chosen in line 3 of the algorithm will satisfy the conclusions of Claim 20.

Conditioned on such $\bar{x}$ and S being chosen, Claims 22 and 23 say that with probability at least $1/2 - o(1) > 1/4$, the output of our algorithm is correct on all but at most $O(\gamma + k^{-0.4})$ fraction of $i \in [k']$. Lifting the conditioning on $\bar{x}$ and S, we conclude that our randomized algorithm computes $(1 - O(\gamma + k^{-0.4}))$-Direct product $f^{k'}$ with probability at least $\epsilon^2/24$, where the probability is over both the input k'-tuple $\bar{x}$ and the internal randomness of the algorithm. By averaging, if we randomly fix the internal randomness used by our algorithm, we get with probability at least $\epsilon' = \epsilon^2/48$ a deterministic circuit that ϵ'-computes $(1 - O(\gamma + k^{-0.4}))$-Direct product $f^{k'}$. The running time bound is obvious. □

4.3 Analysis of algorithm $LEARN$

Here we will prove Lemma 9, which we re-state below. First we recall the definition of a (γ, κ)-good distribution. For parameters k, n, t, let D be a probability distribution over t blocks where each block consists of $k - 1$ pairs (x, b_x), with $x \in \{0,1\}^n$ and $b_x \in \{0,1\}$. Let D_x be the probability distribution over $\{0,1\}^{n(k-1)t}$ obtained from D by keeping only x from every pair (x, b_x). For parameters $0 \leqslant \kappa, \gamma \leqslant 1$, we say that the distribution D is (γ, κ)-*good* if

1. for every sample from D, each block $(x_1, b_1), \ldots, (x_{k-1}, b_{k-1})$ of the sample is such that $b_j = f(x_j)$ for at least $(1 - \gamma)$ fraction of $j \in [k - 1]$, and
2. the distribution D_x is statistically κ-close to the uniform distribution over $\{0,1\}^{n(k-1)t}$.

Lemma 9 [Analysis of Algorithm $LEARN$]. *For any $\mu \geqslant \nu > 0$, $\kappa > 0$, $\epsilon > e^{-k^{\nu/3}}$, $\rho = \frac{\log(1/\epsilon)}{k} + k^{-\nu/2}$, and $t = O((\log 1/\rho)/\epsilon^2)$, there is a probabilistic algorithm $LEARN$ satisfying the following. Let f be an n-variable Boolean function such that a circuit C ϵ-computes $(1-k^{-\mu})$-Direct Product f^k, and let D be a probability distribution over $(\{0,1\}^n \times \{0,1\})^{(k-1)t}$ that is $(k^{-\nu}, \kappa)$-good. Then algorithm $LEARN$, given as input C and a random sample from D, outputs with probability at least $1 - O(\rho) - \kappa$ a circuit C' that computes f on at least $1 - O(\rho)$ fraction of inputs. The running time of $LEARN$, and hence also the size of the circuit C', is at most* $\text{poly}(|C|, 1/\epsilon)$.

We recall the proof of the Direct Product Lemma from [IW97]. 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 [IW97] 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 $1/2 + \Omega(\epsilon)$ (where $\delta > \Omega((\log 1/\epsilon)/k)$). 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 generalize the argument of [IW97] in two ways. First, we assume that the given circuit C ϵ-computes $(1-\gamma)$-Direct Product f^k. Secondly, 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 samples that come from a $(k^{-\nu}, \kappa)$-good distribution D.

Our modification of the analysis from [IW97] will be as follows. After computing the number z of mistakes the circuit makes on inputs coming from the samples (x, b_x), we will subtract from z γk errors that a circuit computing the $(1-\gamma)$-Direct Product f^k can make (even for the "good" k-tuples) as well as $k^{-\nu}(k-1)$ errors that can be present in our imperfect sample coming from the distribution D. Let $w = z - \gamma k - k^{-\nu}(k-1)$. We will decide with probability λ^w to believe the output of the circuit on our input of interest, and with the remaining probability we use a fair coin flip as our prediction for the value of f. The choice of the parameter λ will now depend on the amount of extra errors we have to deal with. It turns out sufficient to set $\lambda = (1/2)^{1/(k^{-\nu}(k-1))}$.

More formally, we have the following lemma for the case $(k^{-\nu}, 0)$-good distribution D.

Lemma 24. *For $1 > \mu \geqslant \nu > 0$ and $\epsilon > e^{-k^{\nu/3}}$, let f be an n-variable Boolean function such that a circuit C ϵ-computes $(1-k^{-\mu})$-Direct Product f^k. Let D be a distribution on $(k-1)$-tuples $(x_1, b_1), \ldots, (x_{k-1}, b_{k-1})$ such that*

1. *the x_js are independent uniformly distributed random variables over $\{0,1\}^n$, and*
2. *for every sample from D, we have $b_j = f(x_j)$ for at least $(1-k^{-\nu})$ fraction of $j \in [k-1]$.*

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 at least $\delta = k^{-\nu/2} + \frac{8\ln(100/\epsilon)}{k}$ and sufficiently large k ($k \geqslant k^\nu + 1$),

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

Moreover, a sample from $\mathcal{F}$ can be produced in time $\mathrm{poly}(1/\epsilon, |C|)$*, given input C and one sample from D.*

Proof. It will be convenient for us to view the distribution D as a pair of distributions $(U, B|_U)$, where U is uniform over $\{0,1\}^{n(k-1)}$ (think of them as $k-1$ strings $y_1, \ldots, y_{k-1}$) and $B|_U$ is the distribution on $\{0,1\}^{k-1}$ conditioned on U. So sampling $(y_1, b_1), \ldots, (y_{k-1}, b_{k-1})$ from D is equivalent to sampling $y_1, \ldots, y_{k-1}$ uniformly at random, and then sampling $b_1, \ldots, b_{k-1}$ from B conditioned on $y_1, \ldots, y_{k-1}$. By our assumption, every sample from B has the property that $b_i = f(y_i)$ for all but at most $k^{-\nu}$ fraction of positions $i \in [k-1]$.

Let $\gamma = k^{-\mu}$ and $\gamma' = k^{-\nu}$. Consider the following distribution $\mathcal{F}$ on randomized circuits F: "On input x,

1. sample $i \in [k]$ uniformly at random, and set $x_i = x$;

2. sample $y_1, \ldots, y_{k-1}$ from U, and assign the values $y_1, \ldots, y_{k-1}$ to x_j for $j \in [k] \setminus \{i\}$;

3. evaluate the circuit C on the input $(x_1, \ldots, x_k)$;

4. sample $b_1, \ldots, b_{k-1}$ from the distribution B conditioned on the x_js for $j \in [k] \setminus \{i\}$;

5. let z be the number of indices $j \in [k] \setminus \{i\}$ where the jth output of the circuit C disagrees with the corresponding value b_j; let $slack = \gamma'(k-1) + \gamma k$. If $z \leqslant slack$, then output the ith output of the circuit C. Otherwise, for $w = z - slack$ and $\lambda = 2^{-1/\gamma'(k-1)}$, with probability λ^w output the ith output of the circuit C, and with the remaining probability output a random coin flip."

Let H be any set of density δ. Suppose we pick $x \in H$ uniformly at random, and then sample a random circuit F according to the distribution $\mathcal{F}$ defined above. The random choice of $x \in H$ and random choices of the first two steps of the algorithm for F induce a probability distribution $\mathcal{E}$ on $(i; x_1, \ldots, x_k)$. Let $\mathcal{E}'$ be the marginal probability distribution on $(x_1, \ldots, x_k)$ induced by $\mathcal{E}$. Note that in step 3 of the algorithm described above, we run the circuit C exactly on those k-tuples that come from the distribution $\mathcal{E}'$.

We have the following.

Claim 25. *The probability distribution $\mathcal{E}$ is equivalent to the following distribution: sample a k-tuple $(x_1, \ldots, x_k)$ according to $\mathcal{E}'$, sample $i \in [k]$ uniformly at random among the positions that contain strings from H, and output $(i; x_1, \ldots, x_k)$.*

Proof. Let $N = 2^n$. A string $(i; x_1, \ldots, x_k)$, with $x_i \in H$, has probability exactly $\frac{1}{k}\frac{1}{|H|}\frac{1}{N^{k-1}}$ according to $\mathcal{E}$. On the other hand, $\mathcal{E}'$ assigns to the string $x_1, \ldots, x_k$ the probability $h\frac{1}{k}\frac{1}{|H|}\frac{1}{N^{k-1}}$, where h is the number of positions $j \in [k]$ such that $x_j \in H$. It follows that for given a sample from $\mathcal{E}'$, the value of i is distributed uniformly over the h positions containing strings from H. □

Thanks to Claim 25, we can equivalently view the distribution $\mathcal{F}$ of circuits F on random inputs from H as follows:

1. sample $(x_1, \ldots, x_k)$ from $\mathcal{E}'$;

2. evaluate the circuit C on the input $(x_1, \ldots, x_k)$;

3. sample $i \in [k]$ uniformly at random among those j where $x_j \in H$;

4. sample $b_1, \ldots, b_{k-1}$ from the distribution B conditioned on the x_js for $j \in [k] \setminus \{i\}$;

5. let z be the number of indices $j \in [k] \setminus \{i\}$ where the jth output of the circuit C disagrees with the corresponding value b_j; let $t = \gamma'(k-1) + \gamma k$. If $z \leqslant t$, then output the ith output of the circuit C. Otherwise, for $w = z - t$ and $\lambda = 2^{-1/\gamma'(k-1)}$, with probability λ^w output the ith output of the circuit C, and with the remaining probability output a random coin flip.

In the rest of the proof, we will use the given equivalent description of $\mathcal{F}$ on a random input from H to estimate the probability that the answer obtained in step 5 is correct.

The next two claims show that $\mathcal{E}'$ behaves very similarly to the uniform distribution.

Claim 26. $\mathbf{Pr}_{\bar{x}=(x_1,\ldots,x_k) \leftarrow \mathcal{E}'}[\bar{x}$ *contains fewer than* $\delta k/2$ *elements from* $H] \leqslant \epsilon/200$.

Proof. For a random k-tuple of n-bit strings, the expected number of strings falling in the set H is δk. By Chernoff, the fraction of those k-tuples that contain fewer than $\delta k/2$ strings from H is at most $e^{-\delta k/8}$. By our choice of δ in Lemma 24, this is at most $\epsilon/100$.

Observe that a k-tuple that contains exactly s strings from H is assigned in $\mathcal{E}'$ the probability $\frac{s}{k}\frac{1}{|H|}\frac{1}{2^{n(k-1)}}$, which is exactly $\frac{s}{\delta k}$ times the probability of this k-tuple under the uniform distribution. Hence the collection of k-tuples that contain fewer than $s = \delta k/2$ strings from H is assigned by $\mathcal{E}'$ the weight at most $1/2$ of their weight under the uniform distribution. The claim follows. □

Claim 27. *Distribution $\mathcal{E}'$ assigns weight at least 0.49ϵ to the collection of k-tuples $(x_1, \ldots, x_k)$ such that they contain at least $\delta k/2$ elements from H and $C(x_1, \ldots, x_k)$ and $f^k(x_1, \ldots, x_k)$ agree in at least $(1-\gamma)$ fraction of places.*

Proof. We know that C does well on at least ϵ fraction of k-tuples under the uniform distribution. By Chernoff, all but $\epsilon/100$ fraction of these tuples will have fewer than $\delta k/2$ strings from H. So, under the uniform distribution, for at least 0.99ϵ fraction of k-tuples we have that C is correct on at least $(1-\gamma)$ fraction of places, and that the tuple contains at least $\delta k/2$ strings in H. The latter implies that the collection of such tuples gets in $\mathcal{E}'$ at least $1/2$ of their probability mass under the uniform distribution, which is at least $(1/2)0.99\epsilon \geqslant 0.49\epsilon$. □

We now estimate the probability of getting a correct answer in step 5 of our algorithm for a random k-tuple from $\mathcal{E}'$. We will only consider those k-tuples that contain at least $\delta k/2$ elements from H. By Claim 26, this may introduce an additive error of at most $\epsilon/200$.

We divide the k-tuples containing at least $\delta k/2$ of strings from H into two sets: G_{few} containing k-tuples where C is wrong in at most γk positions, and G_{many} containing the remaining k-tuples. We estimate the success probability of our algorithm separately on G_{few} and G_{many}. In fact, it will be more convenient for us to estimate the *advantage* of our algorithm, i.e., the probability of getting a correct answer in step 5 minus the probability of getting a wrong answer.

Claim 28. *For every k-tuple from G_{few}, the advantage of our algorithm is at least 0.9.*

Proof. Suppose that $\bar{x}$ is a k-tuple from G_{few} which contains $h \geqslant \delta k/2$ strings in H. Suppose that $r \leqslant \gamma k$ is the total number of mistakes the circuit C makes on this tuple, and $l \leqslant r$ is the number of mistakes C makes on elements from H in the tuple.

Let $i \in [k]$ be uniformly chosen among the h positions that contain a string in H. By the assumption on the probability distribution B, we know that the number z of disagreements between the jth outputs of C, for $j \neq i$, and the corresponding bits b_j supplied by B is at most $\gamma'(k-1)+\gamma k$. Hence, in step 5 we will output the ith output of C with probability 1. The probability (over the choice of i) that this output is wrong is l/h, and so the advantage is $1 - 2l/h$. Note that $l/h \leqslant 2\gamma/\delta \leqslant 2k^{-\mu}/k^{-\nu/2} \leqslant 2/k^{\mu/2} \leqslant o(1)$. Hence, the advantage is at least $1 - o(1) \geqslant 0.9$, as claimed. □

Claim 29. *For every k-tuple from G_{many}, the advantage of our algorithm is at least $-2^{-\Omega(k^{\nu/2})}$.*

Proof. Suppose a given k-tuple $\bar{x}$ from G_{many} contains $h \geqslant \delta k/2$ elements from H. Suppose that C is incorrect in $r > \gamma k$ positions, and $l \leqslant r$ is the number of mistakes C makes in positions containing strings from H.

Let $i \in [k]$ be uniformly chosen among the h positions that contain a string in H. Let z denote the number of disagreements between the jth outputs of C, for $j \neq i$, and the corresponding bits

b_j supplied by B. By our assumption on the distribution B, we have

$$r - 1 - \gamma'(k-1) \leqslant z \leqslant r + \gamma'(k-1)$$

for every choice of i. So for $w = z - t$, we have

$$r - 1 - 2\gamma'(k-1) - \gamma k \leqslant w \leqslant r - \gamma k.$$

With probability $1 - l/h$, the index i is such that C is correct on x_i. In this case, the advantage of our algorithm is

$$\lambda^w \geqslant \lambda^{r-\gamma k}$$

(since $\lambda \leqslant 1$). With probability l/h, the index i is such that C is wrong on x_i. In this case, the advantage is

$$-\lambda^w \geqslant -\lambda^{r-1-2\gamma'(k-1)-\gamma k}.$$

So overall, the advantage is at least

$$(1 - l/h)\lambda^{r-\gamma k} - (l/h)\lambda^{r-1-2\gamma'(k-1)-\gamma k}. \tag{12}$$

To lowerbound the quantity in Eq. (12), we consider two cases.

- **Case 1:** $r - \gamma k \leqslant 2\gamma'(k-1) + 1$. Then the quantity in Eq. (12) is at least

  $$(1 - l/h)\lambda^{2\gamma'(k-1)+1} - (l/h) \geqslant (1 - l/h)\lambda^{3\gamma'(k-1)} - (l/h),$$

  which is at least $1/8 - 9/8(l/h)$, since $\lambda^{\gamma'(k-1)} = 1/2$. Note that $l \leqslant r \leqslant 2\gamma'(k-1) + 1 + \gamma k$ (due to the assumption of Case 1). Also recall that $h \geqslant \delta k/2 > \sqrt{\gamma'}k/2$. Hence, $l/h \leqslant (2\gamma' + \gamma + 1/k)/(\sqrt{\gamma'}/2) \leqslant (3\gamma' + 1/k)/(\sqrt{\gamma'}/2) \leqslant 8\sqrt{\gamma'} \leqslant o(1)$. It follows that the quantity in Eq. (12) is at least $1/8 - o(1) > 0$ in this case.

- **Case 2:** $r - \gamma k > 2\gamma'(k-1) + 1$. Then the quantity in Eq. (12) is at least

  $$\lambda^{r-\gamma k}((1 - l/h) - (l/h)\lambda^{-3\gamma'(k-1)}), \tag{13}$$

  where we upperbounded $2\gamma'(k-1) + 1$ by $3\gamma'(k-1)$ (which is correct for sufficiently large k). Since $\lambda^{\gamma'(k-1)} = 1/2$, the expression in Eq. (13) is at least $\lambda^{r-\gamma k}(1 - 9(l/h))$.

  If $l < h/9$, this expression is positive. If $l \geqslant h/9$, this expression is at least

  $$-8\lambda^{r-\gamma k} \geqslant -8\lambda^{l-\gamma k} \geqslant -8\lambda^{h/9-\gamma k} \geqslant -8\lambda^{\delta k/18-\gamma k} \geqslant -8\lambda^{\sqrt{\gamma'}k/18-\gamma' k}.$$

  This can be lowerbounded by $-8(1/2)^{(1/(18\sqrt{\gamma'}))-1}$, again using the fact that $\lambda^{\gamma'(k-1)} = 1/2$. Recalling that $\gamma' = k^{-\nu}$, we conclude that the advantage in this case is at least $-2^{-\Omega(k^{\nu/2})}$.

So in both cases, the quantity in Eq. (12) is at least $-2^{-\Omega(k^{\nu/2})}$, as claimed. □

Finally, we have by Claims 27 and 28 that the advantage due to tuples in G_{few} is at least $(0.49)\epsilon(0.9) > 0.44\epsilon$. By Claim 29, the advantage due to tuples in G_{many} is at least $-2^{-\Omega(k^{\nu/2})}$. By Claim 26, the advantage due to tuples outside of $G_{few} \cup G_{many}$ is at least $-\epsilon/200$. Thus the overall advantage is at least $0.44\epsilon - 2^{-\Omega(k^{\nu/2})} - \epsilon/200$, which is at least 0.43ϵ when $\epsilon > e^{-k^{\nu/3}}$. This means that the success probability of a random circuit F from $\mathcal{F}$ on a random $x \in H$ is at least $1/2 + 0.2\epsilon$, as claimed. □

Proof of Lemma 9. First we argue the case of $\kappa = 0$. By Lemma 24, we get for a 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 $O(\rho)$.

Consider any fixed string $x \notin Bad$. Let M_t denote the Majority circuit applied to t random circuits sampled from $\mathcal{F}$ on input x. By Chernoff, a random circuit M_t is correct on x with probability at least $1 - \rho^2$. By averaging, we get that for at least $1 - \rho$ fraction of circuits M_t, it is the case that M_t is correct on at least $1 - \rho$ fraction of inputs $x \notin Bad$. It follows that if we pick one such circuit M_t at random, then with probability at least $1 - \rho$ it will be correct on all but ρ fraction of inputs $x \notin Bad$. So it will compute the function f on at least $1 - O(\rho)$ fraction of all inputs.

For $\kappa \neq 0$, we will argue that the probability that the algorithm described in the previous paragraph succeeds in finding a good circuit for f can decrease by at most κ. Let us suppose, for contradiction, that this success probability decreases by more than κ. The distribution D can be viewed as a pair (D', D''), where D'' is a distribution on t blocks of $(k-1)$-tuples $(b_1, \ldots, b_{k-1})$, for $b_j \in \{0,1\}$, conditioned on a given sample from D'. Then the following is a statistical test distinguishing between D' and the uniform distribution: Given a sample a of t blocks of $(k-1)$-tuples $x_1, \ldots, x_{k-1}$, 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 $1 - O(\rho)$ fraction of inputs, then accept; otherwise reject.

It follows that the described test accepts uniform a with probability at least $1-\rho$, while it accepts a from D' with probability less than $1-\rho-\kappa$. Hence the two distributions have statistical distance more than κ. A contradiction. □

5 Applications

5.1 Local approximate list-decoding of truncated Hadamard codes

Recall the definition of truncated Hadamard codes. For given $n, k \in \mathbb{N}$, 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 [GL89], we get an efficient approximate list-decoding algorithm for k-truncated Hadamard codes.

Theorem 30. *There is a randomized algorithm A with the following property. Let msg be any $N = 2^n$-bit string. Suppose that the truth table of the Boolean function computed by a circuit C agrees with the k-truncated Hadamard encoding $code_{msg}$ of msg in at least $1/2 + \epsilon$ fraction of positions, where $\epsilon = \Omega(\mathrm{poly}(1/k))$. Then the algorithm A, given C, outputs with probability at least $\mathrm{poly}(\epsilon)$ a Boolean circuit C' such that the truth table of the Boolean function computed by C' agrees with msg in at least $1 - \rho$ fraction of positions, where $\rho = O(k^{-0.1})$. The running time of the algorithm A, and hence also the size of C', is at most $\mathrm{poly}(|C|, 1/\epsilon)$.*

First we prove a more uniform version of Yao's XOR Lemma. Recall that a Boolean function f is *δ-hard with respect to circuit size s* if, for every Boolean circuit C of size at most s, C correctly computes f on at most $1 - \delta$ fraction of inputs. Yao's XOR Lemma [Yao82] says that if a Boolean

function f is δ-hard with respect to circuit size s, then the function $f^{\oplus^k}(x_1, \ldots, x_k) = \oplus_{i=1}^k f(x_i)$ is $1/2 - \epsilon$-hard with respect to circuit size s', where $\epsilon \approx (1-\delta)^k/2$ and $s' = s * \text{poly}(\epsilon, \delta)$.

It is easy to prove Yao's XOR Lemma given the Direct Product Lemma, using the result of Goldreich and Levin [GL89] on list-decoding Hadamard codes. Applying this to our version of the Direct Product Lemma (Theorem 1), we prove the following version of the XOR Lemma.

Lemma 31 (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 $f^{\oplus^k}(x_1, \ldots, x_k)$ on at least $1/2+\epsilon$ fraction of inputs, where each $x_i \in \{0,1\}^n$, and $\epsilon = \Omega(\text{poly}(1/k))$. Then the algorithm A, given C, outputs with probability at least $\epsilon' = \text{poly}(\epsilon)$ 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* $\text{poly}(|C|, 1/\epsilon)$.

For the proof, we will need the following result due to Goldreich and Levin.

Lemma 32 ([GL89])**.** *There is a probabilistic algorithm A with the following property. Let $x \in \{0,1\}^n$ be any string, and let $B : \{0,1\}^n \to \{0,1\}$ be any predicate such that $\mathbf{Pr}_{r \in \{0,1\}^n}[B(r) = \langle x, r\rangle] \geqslant 1/2 + \gamma$, for some $\gamma > 0$. Then, given oracle access to B, the algorithm A runs in time* $\text{poly}(n, 1/\gamma)$*, and outputs the string x with probability at least $\Omega(\gamma^2)$.*

Proof of Lemma 31. Consider the function $F : \{0,1\}^{2nk} \times \{0,1\}^{2k} \to \{0,1\}$ defined as follows: For $x_1, \ldots, x_{2k} \in \{0,1\}^n$ and $r \in \{0,1\}^{2k}$,

$$F(x_1, \ldots, x_{2k}, r) = \langle f(x_1) \ldots f(x_{2k}), r\rangle.$$

Note that conditioned on $r \in \{0,1\}^{2k}$ having exactly k 1s, the function $F(x_1, \ldots, x_{2k}, r)$ is distributed exactly like the function $f^{\oplus^k}(x_1, \ldots, x_k)$, for uniformly and independently chosen x_is.

Consider the following algorithm for computing F. Given an input $x_1, \ldots, x_{2k}, r$, count the number of 1s in the string r. If it is not equal to k, then output a random coin flip and halt. Otherwise, simulate the circuit C on the sub-tuple of $x_1, \ldots, x_{2k}$ of size k which is obtained by restricting $x_1, \ldots, x_{2k}$ to the positions in r that are 1, and output the answer of C.

Let p be the probability that a random $2k$-bit string contains exactly k 1s. It is easy to see that the described algorithm for computing F is correct with probability at least $(1-p)/2 + p(1/2+\epsilon) = 1/2 + p\epsilon$. Since $p \geqslant \Omega(1/\sqrt{k})$, we get that our algorithm for F is correct with probability at least $1/2 + \epsilon'$, for $\epsilon' = \Omega(\epsilon/\sqrt{k})$.

By a Markov-style argument, we have that for each of at least $\epsilon'' = \epsilon'/2$ of the $2k$-tuples $x_1, \ldots, x_{2k}$, our algorithm computes $F(x_1, \ldots, x_{2k}, r)$ for at least $1/2 + \epsilon''$ fraction of rs. Using the Goldreich-Levin algorithm of Lemma 32, we get a randomized algorithm that computes $f^{2k}(x_1, \ldots, x_{2k})$ with probability at least $\epsilon''' = \Omega(\epsilon''^3)$, where the probability is both over the input $2k$-tuples $x_1, \ldots, x_{2k}$ and the internal randomness of our randomized algorithm. By averaging, randomly fixing the internal randomness of the algorithm yields, with probability at least $\epsilon'''/2$, a deterministic circuit $(\epsilon'''/2)$-computing the Direct Product function $f^{2k}(x_1, \ldots, x_{2k})$. Finally, applying the algorithm of Theorem 1 to this Direct Product circuit yields, with probability $\text{poly}(\epsilon)$, a circuit computing f on at least $1-\rho$ fraction of inputs of f, as required. □

Observe that Lemma 31 gives us an algorithm for decoding a version of truncated Hadamard codes where, instead of sets of size k, the codeword is indexed by tuples of size k. Moreover, the decoding is *local* in that, once we compute the list of circuits for the original Boolean function (whose truth table is viewed as the message), we can run each of these circuits on a given input x to produce the xth bit of the original message. Using Lemma 31, we also get an approximate list-decoding algorithm for the original version of truncated Hadamard codes.

Proof of Theorem 30. The proof is by a reduction to Lemma 31. Let $f : \{0,1\}^n \to \{0,1\}$ be the Boolean function with the truth table msg. Given the circuit C from the statement of the theorem, we construct the following new circuit C'' for computing the XOR function $f^{\oplus^k}$: Given a k-tuple $(x_1, \ldots, x_k)$ of n-bit strings, check if they are all distinct. If so, then run the circuit C on the set $\{x_1, \ldots, x_k\}$, outputting the answer of C. Otherwise, output a random coin flip.

Let p be the probability that a random k-tuple of n-bit strings contains more than one occurrence of some string. Then the described circuit C'' is correct on at least $p/2 + (1-p)(1/2+\epsilon) = 1/2 + (1-p)\epsilon$ fraction of inputs. For $k = \text{poly}(n)$ (or even larger k), it is easy to upperbound p by $2^{-\Omega(n)}$. So the algorithm C'' is correct on at least $1/2 + \epsilon/2$ fraction of inputs.

Running the algorithm of Lemma 31 on the constructed circuit C'' gives us, with probability at least $\text{poly}(\epsilon)$, a circuit C' that $(1-\rho)$-computes the function f. □

5.2 Uniform hardness amplification in $\mathsf{P}^{\mathsf{NP}_\parallel}$

Here we will prove Theorem 2. First we recall some definitions. 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 [TV02]: the advice may depend on the internal randomness of the algorithm but is independent of the given input.

Our Theorem 30 (or rather its version for the XOR Lemma, Lemma 31) immediately gives us hardness amplification for probabilistic algorithms with small amount of advice.

Lemma 33. *Suppose $f : \{0,1\}^n \to \{0,1\}$ is a 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 (randomness dependent) 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.*

First, we observe that our Lemma 31 immediately gives us hardness amplification in the nonuniform setting with very small amount of advice.

Lemma 34. *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function family. If there is a probabilistic polynomial-time algorithm that agrees with the function $f^{\oplus^k}$ on at least $1/2+\epsilon$ fraction of inputs, for some $k = \text{poly}(n)$ and $\epsilon = \text{poly}(1/k)$, then there is a probabilistic polynomial-time algorithm that, given advice of size $O(\log 1/\epsilon)$, agrees with f on at least $1-\delta$ fraction of inputs, for $\delta \leqslant O(k^{-0.1})$.*

Proof. On an input x, we do the following. Given a probabilistic algorithm $(1/2+\epsilon)$-computing the XOR function $f^{\oplus^k}$, we apply to it the algorithm of Lemma 31 for some $\text{poly}(1/\epsilon)$ number times. This gives us a list of $\text{poly}(1/\epsilon)$ 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 $\log \text{poly}(1/\epsilon) = O(\log 1/\epsilon)$ can then be used to identify the correct circuit on the list. We output the result of running this circuit on the input x. □

Now Lemma 33 is an immediate corollary of Lemma 34 above.

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 [BDCGL92] to obtain the following lemma.

Lemma 35. *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.*

We will need the average-case "search-to-decision" reduction for NP from [BDCGL92]; we state this result in the form it was stated in [Tre05].

Lemma 36 ([BDCGL92])**.** *Let $L \in \mathsf{NP}$ be any language, and let $R(\cdot,\cdot)$ be the polynomial-time relation defining L, where $|y| \leqslant w(|x|)$ for some polynomial function $w(\cdot)$. Then there exist a language $L' \in \mathsf{NP}$, a polynomial $l(\cdot)$, and a probabilistic polynomial-time algorithm A such that the following holds. Given a circuit C' that $(1-\delta')$-computes L' on inputs of length $l(n)$, the algorithm A outputs, with probability at least $1 - 2^{-\mathrm{poly}(n)}$, a circuit C that solves the search version of L (with respect to R) on at least $1-\delta$ fraction of inputs of size n, for $\delta = O(\delta' w^2(n))$.*

Proof of Lemma 35. Let L be a given $\delta = 1/n^c$-hard language in NP, let R be its defining relation, and let $w(n) = n^a$ be the upper-bound on the witness size for n-bit inputs in L. We claim that the language $L' \in \mathsf{NP}$ given in Lemma 36 is $\delta' = \Omega(\delta/w^2(n))$-hard, on $l(n)$-bit inputs, with respect to probabilistic polynomial-time algorithms with $O(\log n)$ bits of advice.

Indeed, suppose there is an advice-taking probabilistic polynomial-time algorithm that $(1-\delta')$-computes L'. Enumerate all polynomially many advice strings used by this algorithm, getting a list of polynomially many circuits such that, with probability exponentially close to 1, at least one of the circuits on the list will $(1-\delta')$-compute L'. Apply the probabilistic polynomial-time algorithm A of Lemma 36 to each of the circuits on the list. This yields a list of circuits such that, with high probability, at least one of them solves the search version of L (with respect to R) for at least $1-\delta$ fraction of inputs. Simulate each of these circuits on a given input x, and accept iff at least one of them produces a witness y such that $R(x,y)$ holds. It follows that we have a probabilistic polynomial-time algorithm $(1-\delta)$-computing L, contradicting the assumed hardness of L. □

Combining Lemma 33 and Lemma 35, we obtain the following.

Theorem 37. *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 $g \in \mathsf{P}^{\mathsf{NP}_\parallel}$ 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 $\mathsf{P}^{\mathsf{NP}_\parallel}$ implies the existence of a hard function in NP, and so Theorem 37 can be used to achieve uniform hardness amplification in $\mathsf{P}^{\mathsf{NP}_\parallel}$ claimed in Theorem 2.

Proof of Theorem 2. Let f be computed by a SAT-oracle Turing machine M in time n^e, for some constant e. Define a new Boolean function h as follows: For $x \in \{0,1\}^n$ and $i \in [n^e]$, $h(x,i) = 1$ iff M on input x makes at least i SAT-oracle queries and the ith SAT-oracle query is a satisfiable formula.

Clearly, the function h is in NP: Given x and i, we simulate the oracle machine M on x, recording all SAT-oracle queries asked by M — this can be done since M asks all its queries *in parallel*. If the number of queries is less than i, we reject. Otherwise, we nondeterministically check that the ith SAT-oracle query is a satisfiable formula.

Next we argue that the function h is at least $1/(n^c n^e)$-hard with respect to probabilistic polynomial-time algorithms. Indeed, suppose this is not the case. Then there is a probabilistic polynomial-time

algorithm A such that for each of at least $1 - 1/n^c$ fraction of xs the algorithm A errs on less than $1/n^e$ fraction of inputs (x, i). Since, for each such x, the number of different inputs (x, i) is at most n^e, we conclude that A is correct on all SAT-oracle queries made by the machine M on input x. Hence, using this algorithm A to answer SAT-oracle queries, we get a probabilistic polynomial-time algorithm $(1 - 1/n^c)$-computing f, which contradicts the assumed hardness of f.

Applying Theorem 37 to the NP-function h, we get the required hard function $g \in \mathsf{P}^{\mathsf{NP}_\parallel}$. □

Trevisan [Tre05] gives uniform hardness amplification for NP: If NP contains a language that is $1/\mathrm{poly}(n)$-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 2 achieves much better hardness amplification: from $1/\mathrm{poly}(n)$-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.

Remark 38. It is also possible to prove a "scaled-up" version of Theorem 2 where the hardness is against $2^{n^{o(1)}}$-time probabilistic algorithms, and the amplification goes from hardness $1/n^c$ to $1/2 - 2^{-n^{o(1)}}$.

6 Concluding remarks

The direct product is a generic construction widely used in complexity theory and coding theory. For instance, Alon et el. [ABN+92] used (a derandomized version of) the direct product construction to obtain error-correcting codes with large distance from codes with small distance; rather than picking all possible k-tuples of bit positions of a given n-bit message, they use the neighbors of vertices in a k-regular expander graph on n vertices as their k-tuples. The simplicity of this purely combinatorial, expander-based construction allowed for the subsequent development of linear-time encodable and list-decodable codes [GI03]; this linear-time efficiency appears to be impossible to achieve using algebra-based codes.

A derandomized direct product construction similar in spirit to the construction of [ABN+92] was used in a recent breakthrough result of Dinur [Din06] as a way to amplify the "unsatisfiability" of unsatisfiable cnf formulas, which led to a significantly simpler new proof of the famous PCP Theorem. The direct product part in Dinur's construction is to ask that every vertex v of a graph (an instance of the graph colorability problem) must give not only its own color, but also the colors of all of its neighbors within some constant distance from v (in the graph obtained by adding edges of an expander to those of the original graph); this is similar to asking for the value of a hard function not just on one input but on several inputs simultaneously, thereby amplifying the hardness of the function. Intuitively, asking for such an extended coloring makes it easier to spot an improperly colored edge, thereby amplifying the "non-colorability" of the original graph. Again, the transparent combinatorial nature of the direct product construction played a major role in getting this simple proof of the PCP Theorem.

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.

The obvious open question is to strengthen the parameters of the decoding algorithm. Ideally, we would like to prove the following *dream version* of our Main Theorem:

Dream Version of Direct Product Decoding. *There is a randomized algorithm A that, given a circuit C ϵ-computing the direct product f^k of an n-variable Boolean function f, outputs with probability at least* $\mathrm{poly}(\epsilon)$ *a circuit C' that computes f on all but at most δ fraction of inputs for $\delta \leqslant O((\ln 1/\epsilon)/k)$. The running time of A is* $\mathrm{poly}(|C|/\epsilon)$.

In the present paper, the bottleneck is in our Direct Product amplification procedure. Every step of amplification costs a quadratic decrease in the fraction of tuples where the new circuit (approximately) computes a larger direct product. Is there a more efficient way to do Direct Product amplification?

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 has been pursued in [BOKS06], where Trevisan's amplification results for NP [Tre03, Tre05] are re-proved by combining decoding algorithms for certain monotone codes and the decoding algorithm for the direct-product code from the present paper. Improving Trevisan's results seems to require efficient list-decoding algorithms for monotone codes.

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 [Raz98] can be used in the proof of our Sampling Lemma (Lemma 11). The third author 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. The third author gratefully acknowledges the support by an NSERC Discovery grant.

References

[ABN+92] 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.

[ALM+98] 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).

[AS98] 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).

[BDCGL92] 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.

[BFNW93] 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.

[Bli86] V.M. Blinovsky. Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, 22(1):7–19, 1986.

[BOKS06] J. Buresh-Oppenheim, V. Kabanets, and R. Santhanam. Uniform hardness amplification in NP via monotone codes. *Electronic Colloquium on Computational Complexity*, TR06-154, 2006.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[Din06] 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.

[GI01] 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.

[GI02] 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.

[GI03] 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.

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

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

[GV05] V. Guruswami and S.P. Vadhan. A lower boound on list size for list decoding. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM'05)*, pages 318–329, 2005.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Journal*, pages 13–30, 1963.

[HVV04] 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.

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

[Imp02] R. Impagliazzo. Hardness as randomness: A survey of universal derandomization. *Proceedings of the ICM*, 3:659–672, 2002. (available online at arxiv.org/abs/cs.CC/0304040).

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

[IW01] 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).

[JLR00] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. John Wiley & Sons, Inc., New York, 2000.

[Jus72] J. Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18:652–656, 1972.

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

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

[O'D04] R. O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004. (preliminary version in STOC'02).

[Raz98] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

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

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

[Tre01] L. Trevisan. Extractors and pseudorandom generators. *Journal of the Association for Computing Machinery*, 48(4):860–879, 2001. (preliminary version in STOC'99).

[Tre03] 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.

[Tre04] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[Tre05] 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.

[TV02] 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.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003. (preliminary version in STOC'02).

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

A Combinatorial list-size bounds

Here we will show that, for the codes based on Yao's XOR Lemma, truncated Hadamard codes, and direct-product codes, the combinatorial bound on the list size for radius $(1/2-\epsilon)$ is essentially $\Theta(1/\epsilon^2)$, which is also the upper bound for standard Hadamard codes (see, e.g., [GL89]).

Recall that Yao's XOR Lemma defines the following error-correcting codes. For parameters $n, k \in \mathbb{N}$, a message $msg \in \{0,1\}^n$ is encoded by a codeword $code_{msg} \in \{0,1\}^{n^k}$, where the positions of $code_{msg}$ are indexed by k-tuples $s = (i_1, \ldots, i_k) \in [n]^k$ and $code_{msg}(s) = code_{msg}(i_1, \ldots, i_k) = \oplus_{j=1}^{k} msg(i_j)$. Let us call these codes (n,k)-*XOR-codes*.

Also recall that the k-truncated Hadamard code encodes a message $msg \in \{0,1\}^n$ by a codeword of length $\binom{n}{k}$, which can be indexed by k-size subsets $s \subseteq [n]$ so that the value of the codeword at position $s = \{i_1, \ldots, i_k\}$ is $\oplus_{j=1}^{k} msg(i_j)$.

Observe that the only difference between the XOR-code and truncated Hadamard code is that we use k-tuples of positions of a given n-bit message in the case of XOR-code, and k-size subsets of positions in the case of truncated Hadamard code. For $k \ll n$, a random k-tuple of elements from $[n]$ is very unlikely to contain any repetitions, and so the difference between the two kinds of codes becomes negligible.

We will show that the XOR-code is approximately list-decodable for the choice of parameters assumed in Yao's XOR Lemma. The cases of truncated Hadamard codes and direct product codes are shown similarly.

First, we formally define the notion of approximate list-decodability that we will use. Let $Ham(x,y)$ denote the Hamming distance between binary strings x and y of the same length, i.e., $Ham(x,y)$ is the number of positions where x and y differ.

Definition 39 (Combinatorial approximate list-decodability)**.** We say that a code $code : \{0,1\}^n \to \{0,1\}^m$ is combinatorially δ-approximately (ϵ, t)-list-decodable if for every word $w \in \{0,1\}^m$ there exists a set of at most t strings $msg_1, \ldots, msg_t \in \{0,1\}^n$ satisfying the following: for every $msg \in \{0,1\}^n$, if $code(msg)$ agrees with w in at least $1/2 + \epsilon$ fraction of positions, then there is an $i \in [t]$ such that $Ham(msg, msg_i) \leqslant \delta n$.

Note that for $\delta = 0$, the definition above is equivalent to the more standard notion of (ϵ, t)-list-decodability, saying that any Hamming ball of radius at most $(1/2-\epsilon)$ in $\{0,1\}^m$ contains at most t codewords. In the case of *approximate* list-decodability, we basically say that the Hamming ball of radius $(1/2-\epsilon)$ around any $w \in \{0,1\}^m$ contains at most t codewords $code(msg_1), \ldots, code(msg_t)$ such that any other codeword $code(msg)$ in the same Hamming ball has its message msg within the Hamming distance δn of one of the msg_i's, for $1 \leqslant i \leqslant t$.

Remark 40. We may also assume, without loss of generality, that the set of t messages $msg_1, \ldots, msg_t$ in Definition 39 above is such that $Ham(msg_i, msg_j) > \delta n$, for every $i \neq j \in [t]$; otherwise we can make the requisite set smaller by removing any msg_i that is close to some msg_j, for $j \neq i$.

A.1 Combinatorial approximate list-decodability

A.1.1 XOR-codes

We now show that XOR-codes are approximately list-decodable for an appropriate choice of parameters.

Theorem 41. *For any $\epsilon > 0$ and $\delta > (\ln 1/\epsilon)/k$, the (n,k)-XOR-code is δ-approximately (ϵ, t)-list-decodable for $t = 1/(4\epsilon^2 - e^{-2\delta k}) \leqslant O(1/\epsilon^2)$.*

The proof of Theorem 41 will follow from the next lemma. Below we think of the (n,k)-XOR-code as encoding a message m into a function $code_m : [n^k] \to \{0,1\}$ so that, for any $s \in [n^k]$, the value of $code_m(s)$ is the sth bit of the XOR-encoding of the message m.

Lemma 42. *Let $m_1, \ldots, m_t$ be any n-bit strings satisfying the following two conditions:*

1. *for every two distinct $i, j \in [t]$, m_i and m_j differ in at least δn positions, and*
2. *there is some function $B : [n^k] \to \{0,1\}$ such that, for each $i \in [t]$, $\mathbf{Pr}_{s \in [n]^k}[code_{m_i}(s) = B(s)] \geqslant 1/2 + \epsilon$, where code is the (n,k)-XOR-code.*

Then

$$t \leqslant \frac{1}{4\epsilon^2 - e^{-2\delta k}}.$$

Proof. For every $s \in [n]^k$, let

$$\epsilon_s = \mathbf{Pr}_{i \in [t]}[code_{m_i}(s) = B(s)] - \mathbf{Pr}_{i \in [t]}[code_{m_i}(s) \neq B(s)] = \frac{1}{t} \sum_{i \in [t]} (-1)^{code_{m_i}(s) \oplus B(s)}.$$

Observe that $\mathbf{Exp}_{s \in [n]^k}[\epsilon_s] \geqslant 2\epsilon$. So, we get that $4\epsilon^2 \leqslant (\mathbf{Exp}_s[\epsilon_s])^2$. By Jensen's inequality, the latter is at most $\mathbf{Exp}_s[(\epsilon_s)^2]$.

We have

$$\begin{aligned}
\mathbf{Exp}_s[(\epsilon_s)^2] &= \mathbf{Exp}_s\left[\frac{1}{t^2} \sum_{i,j} (-1)^{code_{m_i}(s) \oplus code_{m_j}(s)}\right] \\
&= \mathbf{Exp}_s\left[\frac{1}{t^2} \sum_{i,j} (-1)^{code_{m_i \oplus m_j}(s)}\right] \\
&= \frac{1}{t^2} \cdot \mathbf{Exp}_s\left[\sum_i (-1)^0 + \sum_{i \neq j} (-1)^{code_{m_i \oplus m_j}(s)}\right] \\
&= \frac{1}{t} + \frac{1}{t^2} \cdot \sum_{i \neq j} \mathbf{Exp}_s\left[(-1)^{code_{m_i \oplus m_j}(s)}\right].
\end{aligned}$$

Next we bound the quantity $\mathbf{Exp}_s\left[(-1)^{code_{m_i \oplus m_j}(s)}\right]$ in the expression above.

Claim 43. *For any $i \neq j$, we have $\mathbf{Exp}_s\left[(-1)^{code_{m_i \oplus m_j}(s)}\right] \leqslant (1 - 2\delta)^k$.*

Proof. First, observe that by the assumption on pairwise distance between messages, we have that the string $m' = m_i \oplus m_j$ has relative Hamming weight $w \geqslant \delta$. We need to compute the probability that a random k-tuple $s \in [n]^k$ hits an even number of 1s in the string m' minus the probability that it hits an odd number of 1s. This is exactly

$$\begin{aligned}\sum_{\text{even } i \in [k]} \binom{k}{i} w^i (1-w)^{k-i} - \sum_{\text{odd } i \in [k]} \binom{k}{i} w^i (1-w)^{k-i} &= \sum_{i \in [k]} \binom{k}{i} (-w)^i (1-w)^{k-i} \\ &= (1-2w)^k.\end{aligned}$$

The latter is at most $(1-2\delta)^k$. □

Using the bound from Claim 43, we have

$$\mathbf{Exp}_s(\epsilon_s)^2 \leqslant \frac{1}{t} + \frac{t(t-1)}{t^2}(1-2\delta)^k \leqslant \frac{1}{t} + e^{-2\delta k}.$$

Recalling that $\mathbf{Exp}_s(\epsilon_s)^2 \geqslant 4\epsilon^2$, we obtain

$$t \leqslant \frac{1}{4\epsilon^2 - e^{-2\delta k}},$$

as required. □

Proof of Theorem 41. For $m = n^k$, let $w \in \{0,1\}^m$ be any word. Let t' be the smallest number of messages $msg_1, \ldots, msg_{t'}$ satisfying the conditions of Definition 39. As observed in Remark 40 above, we may assume that any pair of distinct msg_i and msg_j have Hamming distance greater than δn; otherwise, t' could be made smaller. Then Lemma 42 gives the required upper bound on t'. □

A.1.2 Truncated Hadamard codes

Now we prove essentially the same upper bound on the list size for k-truncated Hadamard codes.

Theorem 44. *For any $\epsilon > 0$ and $\delta > \Omega((\ln 1/\epsilon)/k)$, the k-truncated Hadamard code is δ-approximately (ϵ, t)-list-decodable for $t = 1/(4\epsilon^2 - e^{-0.9\delta k}) \leqslant O(1/\epsilon^2)$.*

The proof of Theorem 44 follows from the next lemma.

Lemma 45. *Let $m_1, \ldots, m_t$ be t n-bit strings satisfying the following two conditions:*

1. *for every two distinct $i, j \in [t]$, m_i and m_j differ in at least δn positions, and*
2. *for each $i \in [t]$, $\mathbf{Pr}_{k\text{-set } s \subseteq [n]}[code_{m_i}(s) = B(s)] \geqslant 1/2 + \epsilon$, where code is the k-truncated Hadamard code.*

Then

$$t \leqslant \frac{1}{4\epsilon^2 - e^{-0.9\delta k}}.$$

Proof. The proof proceeds exactly like the proof of Lemma 42 above. We first define ϵ_s for k-size sets $s \subseteq [n]$, and show that the expectation of $(\epsilon_s)^2$ is lower-bounded by $4\epsilon^2$. Next we upper-bound $\mathbf{Exp}_s(\epsilon_s)^2$. The argument is essentially the same as that in the proof of Lemma 42. The main ingredient is the following claim.

Claim 46. *For any $i \neq j$, we have* $\mathbf{Exp}_s[(-1)^{code_{m_i \oplus m_j}(s)}] \leqslant e^{-0.9\delta k}$.

Proof. To upper-bound the given expectation, we consider the following random experiment. Pick uniformly at random a subset $s' \subseteq [n]$ of size $2k$, and for each $i \in s'$ flip a fair coin to decide whether to keep this element i or to discard it. Call the resulting set s''. Note that the expected size of s'' is k. Also note that, conditioned on flipping exactly k heads in the second stage of our random experiment, the resulting set s'' of size k is uniformly distributed among all size k subsets of $[n]$.

If a set s' chosen in the first stage of the random experiment has a nonempty intersection with the set I of positions in $m' = m_i \oplus m_j$ that contain 1s, then the conditional expectation $\mathbf{Exp}_{s''}[(-1)^{code_{m'}(s'')}] = 0$. If, on the other hand, the intersection is empty, then the conditional expectation $\mathbf{Exp}_{s''}[(-1)^{code_{m'}(s'')}] = 1$. Lifting the conditioning on s', we get

$$\mathbf{Exp}_{s''}[(-1)^{code_{m'}(s'')}] = \mathbf{Pr}_{s'}[s' \cap I = \varnothing\ ].$$

By the second property of m_is, we know that m' contains $w \geqslant \delta$ fraction of positions with 1s. Thus the expected size of the intersection between s' and I is $2kw$, and, by Hoeffding, the probability of having the empty intersection is at most $e^{-kw} \leqslant e^{-\delta k}$.

Finally, $\mathbf{Exp}_s[(-1)^{code_{m'}(s)}]$ is equal to the expectation over s'', conditioned on having exactly k heads in $2k$ Bernoulli trials. The latter probability is at least $\Omega(1/\sqrt{k})$. So we conclude that

$$\mathbf{Exp}_s[(-1)^{code_{m'}(s)}] \leqslant \mathbf{Exp}_{s''}[(-1)^{code_{m'}(s'')}]O(\sqrt{k}) \leqslant O(\sqrt{k})e^{-\delta k} \leqslant e^{-0.9\delta k}.$$

□

Then we conclude the proof in the same way as in Lemma 42. □

A.1.3 Direct product codes

The k-wise direct product encoding of a message $msg \in \{0,1\}^n$ is a string of length n^k over the alphabet $\{0,1\}^k$, such that the value of the encoding at position $(i_1, \ldots, i_k) \in [n]^k$ is the k-tuple $(msg_{i_1}, \ldots, msg_{i_k})$.

It is easy to generalize the notion of approximate list-decodability to the case of these direct product codes. Let $\Sigma = \{0,1\}^k$, and let $m = n^k$. Let $code : \{0,1\}^n \to \Sigma^m$ be the k-wise direct product code. We say that this direct product code is *δ-approximately (ϵ, t)-list decodable* if, for every $B \in \Sigma^m$, there exists a collection of at most t messages $msg_1, \ldots, msg_t \in \{0,1\}^n$ such that, for every $code(msg)$ that agrees with B in at least ϵ fraction of positions, there is some $i \in [t]$ such that $Ham(msg_i, msg) \leqslant \delta n$.

We have the following list-decodability result for direct product codes.

Theorem 47. *For any $\epsilon > 0$ and $\delta > \Omega((\ln 1/\epsilon)/k)$, the k-wise direct product code is δ-approximately (ϵ, t)-list decodable for $t \leqslant 1/(\epsilon^2 - e^{-\delta k}) \leqslant O(1/\epsilon^2)$.*

Proof sketch. The proof is very similar to that of Theorem 41. Given t messages (of pairwise Hamming distance at least δn) whose encodings agree with some word B in at least ϵ fraction of positions, we define for every $s \in [n]^k$ the quantity ϵ_s to be the fraction of messages that agree with B in position s. We have that $\mathbf{Exp}_s[\epsilon_s] \geqslant \epsilon$, and by Jensen, $\epsilon^2 \leqslant \mathbf{Exp}_s[(\epsilon_s)^2]$.

To upper bound the latter expectation, we need to upperbound the probability that for two distinct messages m and m' of Hamming distance at least δn, it is the case that $code(m)_s = code(m')_s$ for a random $s \in [n]^k$. The latter happens only if each of the k indices of s falls outside of the part where m and m' differ. Clearly, the probability of this event is at most $(1-\delta)^k \leqslant e^{-\delta k}$. The rest of the argument is the same as in the proof of Theorem 41. □

A.2 Tightness of the $O(1/\epsilon^2)$ list-size bound

In Theorem 41, we showed that for sufficiently large δ (i.e., $\delta > (\ln 1/\epsilon)/k$), the XOR-code is δ-approximately $(\epsilon, O(1/\epsilon^2))$-list-decodable. Here we will argue that this upper bound of $O(1/\epsilon^2)$ is *tight* in general. We will show that, even for constant δ (say, $\delta = 1/5$), the list size can grow quadratically with $1/\epsilon$.

The lower bound $\Omega(1/\epsilon^2)$ on the list size of general list-decodable is known [Bli86, GV05]. In particular, Guruswami and Vadhan [GV05] show the following result for codes over any alphabet; we shall state it just for binary codes.

Theorem 48 ([GV05])**.** *There exist positive constants c and d such that, for all small enough $\epsilon > 0$, the following holds. If $Code$ is $(\epsilon, c/\epsilon^2)$-list decodable code, then $Code$ has at most $(1/\epsilon)^{d\cdot\epsilon^{-2}}$ codewords.*

Note that if $Code$ is (ϵ, t)-list decodable code that encodes n-bit messages, and if ϵ is large enough so that $(1/\epsilon)^{d\cdot\epsilon^{-2}} < 2^n$, then it must be the case that $t > c/\epsilon^2$.

It is possible to get from Theorem 48 the same lower bound for *approximately* list-decodable codes as well. Let $code' : \{0,1\}^{n'} \to \{0,1\}^n$ be any binary code of constant rate and constant relative distance δ, for some $\delta > 0$; that is, $n \leqslant cn'$ for some constant $c > 0$, and any two codewords of $code'$ are at least Hamming distance δn apart (such codes can be shown to exist by the probabilistic method, as well as constructively [Jus72]). Let $code : \{0,1\}^n \to \{0,1\}^m$ be the k-truncated Hadamard code, for $m = \binom{n}{k}$. Define $Code : \{0,1\}^{n'} \to \{0,1\}^m$ to be the composition of $code'$ and $code$, i.e., we first encode an n'-bit message with $code'$ and then encode the resulting codeword with $code$. It is easy to see that if $code$ is δ-approximately (ϵ, t)-list decodable, then $Code$ is (ϵ, t)-list decodable. Theorem 48 then implies that $t \geqslant \Omega(1/\epsilon^2)$ for δ-approximate (ϵ, t)-list decodable $code$.

This way we can get $\Omega(1/\epsilon^2)$ list-size lower bound for all of our approximately decodable codes: XOR, truncated Hadamard, and direct product codes.

The proofs in [Bli86, GV05] work for general codes, but are somewhat involved. Below we give a short proof for the case of k-truncated Hadamard code, and then give an easy reduction to the case of k-XOR code. Also, our lower bound works even for very small ϵ.

Theorem 49. *Let $m = \binom{n}{k}$, let $\epsilon > \max\{m^{-1/256}, 2^{-n/256}\}$, and let $\delta < 1/4$. Let $code : \{0,1\}^n \to \{0,1\}^m$ be the k-truncated Hadamard code. Then there exists a string $B \in \{0,1\}^m$ and $\ell = \Omega(1/\epsilon^2)$ strings $msg_1, \ldots, msg_\ell \in \{0,1\}^n$ such that*

1. *for all distinct $i, j \in [\ell]$, we have $Ham(msg_i, msg_j) \geqslant \delta n$, and*
2. *for all $i \in [\ell]$, we have $Ham(code(msg_i), B) \leqslant (1/2 - \epsilon)m$.*

Proof. Let $t = 2T+1$ be an odd positive integer to be specified later. Pick strings $a_1, \ldots, a_t \in \{0,1\}^n$ uniformly at random. The probability that any two of them are within Hamming distance less than $n/4$ is at most $O(t^2 e^{-(1/4)^2 n/2})$. This probability is less than $o(1/\sqrt{t})$ for $t < e^{n/90}$.

For a given t-tuple $a = (a_1, \ldots, a_t)$, define the function $B_a : \{0,1\}^n \to \{0,1\}$ as follows: for every $r \in \{0,1\}^n$,

$$B_a(r) = Maj_{1 \leq i \leq t} \langle a_i, r \rangle.$$

We will show that there exists a such that the function B_a evaluated at all strings r of Hamming weight exactly k agrees in at least $1/2 + \Omega(1/\sqrt{t})$ fraction of places with $code(a_i)$, for at least $\Omega(t)$ of a_i's. This will imply the lemma for $\epsilon = \Omega(1/\sqrt{t})$.

The intuition is as follows. Fix a string r of Hamming weight k. Fix an index $i \in [t]$. Randomly choose all the other a_j for $j \neq i$. For each $j \neq i$, the random variable $\langle a_j, r\rangle$ is a fair coin flip (for a random a_j). For different j's, the corresponding random variables are independent. If we flip $2T$ independent fair coins, we get exactly T heads with probability at least $\Omega(1/\sqrt{T})$. Conditioned on getting exactly T heads, we have that $B(r) = \langle a_i, r\rangle$, i.e., $B(r) = \mathit{code}(a_i)_r$. By averaging, we can argue that if we randomly fix a_j's for $j \neq i$, then (with probability at least $\Omega(1/\sqrt{T})$) we will have at least $\Omega(1/\sqrt{T})$ fraction of strings r (of Hamming weight k) for which we have $B(r) = \mathit{code}(a_i)_r$. For every remaining r, the value $B(r)$ is fixed (because of the fixed a_j's for $j \neq i$), but is independent of a_i. The random variables $\langle a_i, r\rangle$ over these remaining r's are uniformly distributed and *pairwise independent*. Hence, a random choice of a_i is likely to result in about $1/2$ of these random variables being equal to the fixed value $B(r)$ (and by the Chebyshev inequality, we can bound the probability that this value deviates from the expectation). Thus, for random $a_1, \ldots, a_t$, we are likely to get B and $\mathit{code}(a_i)$ agree in about $1/2 + \Omega(1/\sqrt{T})$ fraction of positions. With some extra work, we will show that this happens simultaneously for $\Omega(t)$ different indices $i \in [t]$, which yields many codewords with good agreement with the string B. We give a formal argument next.

Given a t-tuple a, we say that a string r is *balanced for* a if the number of indices i with $\langle a_i, r\rangle = 0$ is either T or $T + 1$.

Claim 50. *There is a constant c such that for at least $c/\sqrt{t}$ fraction of random t-tuples a, there are at least $c/\sqrt{t}$ fraction of strings $r \in \{0,1\}^n$ of Hamming weight k such that each r is balanced for a.*

Proof. Each fixed nonzero r (of Hamming weight k) is balanced for a with probability at least $c'/\sqrt{t}$ over the choice of a random t-tuple a, for some constant c'. Hence, we have

$$\mathbf{Pr}_{a,r}[r \text{ is balanced for } a] \geqslant c'/\sqrt{t},$$

where r is a random n-bit string of Hamming weight k. By averaging, the claim follows for $c = c'/2$. □

For any $i \in [t]$, and any $(t-1)$-tuple $a^{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_t)$ of n-bit strings, we say that a string $r \in \{0,1\}^n$ is *i-balanced for a^{-i}* if the number of $j \in [t] \setminus \{i\}$ with $\langle m_j, r\rangle = 0$ is T.

Claim 51. *Let d be any constant. Suppose $i \in [t]$ and $a^{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_t)$ are such that $d/\sqrt{t}$ fraction of strings $r \in \{0,1\}^n$ of Hamming weight k are i-balanced for a^{-i}. Then for a random $a_i \in \{0,1\}^n$ and the t-tuple $a = (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_t)$, we have*

$$\mathbf{Pr}_{a_i}\left[B_a(r) = \langle a_i, r\rangle \text{ for at least } \frac{1}{2} + \frac{d/4}{\sqrt{t}} \text{ of } r\text{'s}\right] > 1 - O\left(\frac{t}{m}\right),$$

where $m = \binom{n}{k}$.

Proof. For r's that are i-balanced for a^{-i}, we have $B(r) = \langle a_i, r\rangle$ whatever a_i is. Let R be the set of the remaining r's of weight k that are not i-balanced. For each $r \in R$, $B(r)$ is determined, and so is independent of the choice of a_i.

Define random variables X_r where $X_r = 1$ if $< a_i, r >= B(r)$, and $X_r = 0$ otherwise. We get that, for $r \in R$, these random variables X_r are uniformly distributed and pairwise independent, for a random choice of a_i. We expect $X_r = 1$ for half of the r's. By Chebyshev, for any constant d', the probability that there are fewer than $1/2 - d'/\sqrt{t}$ fraction of $r \in R$ with $X_r = 1$ is less than $O(t/m)$.

Finally, observe that if the latter event does not happen for some a_i, then we have that the agreement between $\langle a_i, r\rangle$ and $B(r)$ is at least

$$\frac{d}{\sqrt{t}} + \left(1 - \frac{d}{\sqrt{t}}\right)\left(\frac{1}{2} - \frac{d'}{\sqrt{t}}\right) \geqslant \frac{1}{2} + \frac{(d/2) - d'}{\sqrt{t}}.$$

The latter can be made at least $1/2 + (d/4)/\sqrt{t}$ by choosing $d' = d/4$. □

By Claim 50, there are at least $c/\sqrt{t}$ fraction of a's with $c/\sqrt{t}$ fraction of strings r balanced for a. Fix any such a. Observe that each r that is balanced for a is also i-balanced for $T + 1 > t/2$ values of i. Since we have $c/\sqrt{t}$ fraction of r's balanced for a, we get by a simple averaging argument that there are at least $t/4$ values of i such that each has at least $(c/4)/\sqrt{t}$ fraction of i-balanced r's. Note that this happens with probability at least $c/\sqrt{t}$ over a's.

On the other hand, Claim 51 implies that the probability (over t-tuples a) that there is at least one i with $(c/4)/\sqrt{t}$ fraction of i-balanced r's, such that for this i, the agreement between $\langle a_i, r\rangle$ and $B_a(r)$ is less than $1/2 + (c/16)/\sqrt{t}$ is at most $O(t^2/m)$. For $t = o(m^{1/3})$, this probability is less than $o(1/\sqrt{t})$.

Thus there is at least $c/\sqrt{t} - o(1/\sqrt{t}) \geqslant (c/2)/\sqrt{t}$ fraction of a's such that, for each of these a's, we have that

1. there are at least $t/4$ values of i, each having at least $(c/4)/\sqrt{t}$ fraction of i-balanced r's, and
2. for every such i, the agreement between $\langle a_i, r\rangle$ and $B_a(r)$ is at least $1/2 + (c/16)/\sqrt{t}$.

Recall that for all but $o(1/\sqrt{t})$ fraction of $a = (a_1, \ldots, a_t)$ we have that the pairwise Hamming distance between a_i and a_j is at least δn, for all $i \neq j$ in $[t]$. Therefore, there must exist a choice of $a = (a_1, \ldots, a_t)$ and a subset I of $t/4$ of the i's such that, for each $i \in I$, the agreement between $code(a_i)$ and B_a is at least $1/2 + (c/16)/\sqrt{t}$, and the pairwise Hamming distance between any a_i and a_j, for $i \neq j$, is at least δn. Setting $t = ((c/16)/\epsilon)^2$ concludes the proof. □

Now we reduce the case of XOR codes to the case of truncated Hadamard codes, obtaining the following.

Theorem 52. *Let $m = \binom{n}{k}$, $\epsilon > \max\{m^{-1/256}, 2^{-n/256}\}$, $k^2/n \leqslant o(\epsilon)$, and $\delta < 1/4$. Let $code : \{0,1\}^n \to \{0,1\}^{n^k}$ be the (n,k)-XOR-code. Then there exists a string $B \in \{0,1\}^{n^k}$ and $s = \Omega(1/\epsilon^2)$ messages $msg_1, \ldots, msg_s$ of pairwise Hamming distance at least δn such that the agreement between B and $code(msg_i)$ is at least $1/2 + \Omega(\epsilon)$ for each $i \in [s]$.*

Proof. Recall that the (n,k)-XOR encoding of an n-bit message msg is the sequence of $msg(i_1) \oplus \cdots \oplus msg(i_k)$ over all k-tuples of indices $i_1, \ldots, i_k$ from $[n]$. The fraction of those k-tuples $(i_1, \ldots, i_k)$ that contain some index $j \in [n]$ more than once is at most k^2/n, which is $o(\epsilon)$ by our assumption.

Ignoring the k-tuples with repeats, we can partition the remaining k-tuples into ℓ blocks where each block contains $\binom{n}{k}$ tuples corresponding to distinct k-size subsets of $[n]$. For each such block, the XOR encoding of a given message msg (restricted to the k-tuples in the block) coincides with the k-truncated Hadamard encoding of msg. So, the XOR encoding of msg restricted to the k-tuples without repeats is just a concatenation of ℓ copies of the truncated Hadamard encoding of msg.

By Theorem 49, there is a collection of $\Omega(1/\epsilon^2)$ n-bit messages (pairwise Hamming distance δn apart) and a string B' such that the k-truncated Hadamard encoding of each message agrees with B' in at least $1/2 + \Omega(\epsilon)$ fraction of positions. Let B'' be the string obtained as a concatenation of

ℓ copies of the string B'. It follows that for the same collection of messages, their k-XOR encodings will agree with the string B'' in at least $1/2 + \Omega(\epsilon)$ fraction of positions, when the positions are restricted to the k-tuples without repeats. Let us now pad B'' with enough 0's to get the string of length n^k. Let us call the new string B. We have that the XOR encodings of our messages will agree with B in at least $1/2 + \Omega(\epsilon) - k^2/n \geqslant 1/2 + \Omega(\epsilon)$ fraction of positions. □

A.3 List size for small δ

We showed that for the XOR-code the list size is small when $\delta > (\ln 1/\epsilon)/k$. Next, we show that the list size could be exponentially large when we allow the list to contain messages which are not far apart from one another.

Theorem 53. *Let $\delta \leqslant \min\{(\ln 1/\epsilon)/(4k), 1/17\}$. There exists a set of at least $2^{\delta n - 1}$ of n-bit messages of pairwise Hamming distance at least δn and such that the (n,k)-XOR encoding of each message has agreement at least $1/2 + \epsilon$ with the string 0^{n^k}.*

The proof of this theorem will follow from the two lemmas below.

Lemma 54. *Let $\delta \leqslant \min\{(\ln 1/\epsilon)/(4k), 1/3\}$ and let $m_1, \ldots, m_l$ be n-bit strings of Hamming weight δn. Let code be the (n,k)-XOR-code. Then, for every $i \in [l]$, we have*

$$\mathbf{Pr}_{s \in [n]^k}[code_{m_i}(s) = 0] = 1/2 + 1/2(1-2\delta)^k \geqslant 1/2 + \epsilon.$$

Proof. For a given message m_i, $code_{m_i}(s) = 0$ when s has an even intersection with the subset of positions in m_i which are 1. The probability of this event is exactly $\sum_{\text{even } i \in [k]} \binom{k}{i} \delta^i (1-\delta)^{k-i}$ which is $1/2 + (1-2\delta)^k/2 \geqslant 1/2 + \epsilon$ when $\delta \leqslant \min\{(\ln 1/\epsilon)/(4k), 1/3\}$. □

Lemma 55. *Let $M = \{m_1, \ldots, m_l\}$ be a set of n-bit strings of Hamming weight δn, where $\delta \leqslant 1/17$. Then there is a subset $N \subseteq M$ of size at least $2^{\delta n - 1}$ such that any two messages in N have Hamming distance at least δn.*

Proof. Consider a graph with the messages in M denoting the vertices and there is an edge between two messages if they differ in less than δn positions. Set N to be an independent set of this graph, chosen as follows: pick a vertex v and place it in N, delete v and its neighboring vertices; repeat until no vertices are left.

The size of N is lowerbounded by $|M|/(\Delta + 1)$, where Δ is the maximum degree of the graph. We have

$$\begin{aligned}
\Delta &= \sum_{i=1}^{\delta n/2} \binom{\delta n}{i} \binom{(1-\delta)n}{i} \\
&\leqslant \binom{(1-\delta)n}{\delta n/2} \sum_{i=1}^{\delta n/2} \binom{\delta n}{i} \quad \text{(since } \delta \leqslant 1/2\text{)} \\
&\leqslant \binom{(1-\delta)n}{\delta n/2} \cdot 2^{\delta n} \\
&\leqslant 2^{\delta n} \cdot \binom{n}{\delta n/2}.
\end{aligned}$$

This yields

$$\begin{aligned}
|N| &\geqslant \frac{\binom{n}{\delta n}}{\Delta + 1} \\
&\geqslant \frac{\binom{n}{\delta n}}{2 \cdot 2^{\delta n} \cdot \binom{n}{\delta n/2}} \\
&= \frac{1}{2^{\delta n+1}} \cdot \frac{n(n-1)\ldots(n-\delta n+1)}{n(n-1)\ldots(n-\delta n/2+1)} \cdot \frac{(\delta n/2)!}{(\delta n)!} \\
&= \frac{1}{2^{\delta n+1}} \cdot \frac{(n-\delta n/2)(n-\delta n/2-1)\ldots(n-\delta n+1)}{(\delta n)(\delta n-1)\ldots(\delta n/2+1)} \\
&\geqslant \frac{(1/\delta - 1)^{\delta n/2}}{2^{\delta n+1}} \\
&\geqslant 2^{\delta n-1} \quad (\text{for } \delta \leqslant 1/17).
\end{aligned}$$

□

Proof of Theorem 53. The proof is immediate from Lemma 54 and Lemma 55. □

One also gets a similar lower bound on the list size for k-truncated Hadamard codes and direct product codes. The proof for k-truncated codes is the same as that of Theorem 53 above, except for considering k-size subsets of $[n]$ rather k-tuples in Lemma 54. A version of Lemma 54 continues to hold in the case k-size subsets as well, and can be proved using the ideas from the proof of Claim 46.