Electronic Colloquium on Computational Complexity, Report No. 99 (2019)

Nearly Optimal Pseudorandomness From Hardness

Dean Doron[*]
Department of Computer Science
University of Texas at Austin
deandoron@utexas.edu

Dana Moshkovitz[†]
Department of Computer Science
University of Texas at Austin
danama@cs.utexas.edu

Justin Oh[‡]
Department of Computer Science
University of Texas at Austin
sjo@cs.utexas.edu

David Zuckerman[§]
Department of Computer Science
University of Texas at Austin
diz@utexas.edu

Abstract

Existing proofs that deduce $\mathbf{BPP} = \mathbf{P}$ from circuit lower bounds convert randomized algorithms into deterministic algorithms with a large polynomial slowdown. We convert randomized algorithms into deterministic ones with little slowdown. Specifically, assuming exponential lower bounds against nondeterministic circuits, we convert any randomized algorithm over inputs of length n running in time $t \geq n$ to a deterministic one running in time $t^{2+\alpha}$ for an arbitrarily small constant $\alpha > 0$. Such a slowdown is nearly optimal, as, under complexity-theoretic assumptions, there are problems with an inherent quadratic derandomization slowdown. We also convert any randomized algorithm that errs rarely into a deterministic algorithm having a similar running time (with pre-processing).

Our results follow from a new, nearly optimal, explicit pseudorandom generator fooling circuits of size s with seed length $(1+\alpha)\log s$, under the assumption that there exists a function $f \in \mathbf{E}$ that requires nondeterministic circuits of size at least $2^{(1-\alpha')n}$, where $\alpha = O(\alpha')$. The construction uses, among other ideas, a new connection between pseudoentropy generators and locally list recoverable codes.

[*]Supported by NSF Grant CCF-1705028.
[†]Supported in part by NSF Grant CCF-1705028 and CCF-1648712.
[‡]Supported by NSF Grant CCF-1705028.
[§]Supported in part by NSF Grant CCF-1705028 and a Simons Investigator Award (#409864).

ISSN 1433-8092

Contents

1 Introduction **1**
1.1 Pseudorandom Generators and Derandomization . . . 1
1.2 Beating the Hybrid Argument . . . 3
1.3 Locally Decodable Codes All the Way Down . . . 4
1.4 From Pseudoentropy to Pseudorandom Bits . . . 6
1.5 Open Problems . . . 7

2 Preliminaries **8**
2.1 Circuits, Nondeterministic Circuits and Worst-Case Hardness . . . 8
2.2 Error Correcting Codes . . . 9
2.3 Random Variables, Min-Entropy . . . 10
2.4 Condensers, Extractors and Samplers . . . 11
2.5 Pseudoentropy . . . 13
2.6 Pseudoentropy Generators and Pseudorandom Generators . . . 15

3 A Locally List Recoverable Code From Local List Decoding **15**
3.1 The Construction . . . 16
3.2 Analysis . . . 17

4 A Metric Pseudoentropy Generator From Worst-Case Hardness **20**

5 Derandomizing Algorithms That Err Rarely **22**

6 Pseudoentropy Fooling Real-Valued Distinguishers **23**
6.1 **Metric*** Pseudoentropy . . . 24
6.2 Extracting Randomness From **Metric*** Pseudoentropy . . . 26
6.3 **Yao*** Pseudoentropy . . . 28

7 A Metric* Pseudoentropy Generator From Worst-Case Hardness **29**

8 Extractors With Near-Optimal Seed Length **30**
8.1 Computing the Extractor by a Small Circuit . . . 32

9 PRGs With Nearly Optimal Slowdown **36**
9.1 On Derandomizing AM . . . 38

10 Acknowledgement **40**

A Condensers and List Recoverable Codes **45**

1 Introduction

1.1 Pseudorandom Generators and Derandomization

Randomized algorithms can outperform deterministic algorithms. We know randomized polynomial time algorithms for problems such as polynomial identity testing, factoring polynomials over large fields, and approximating the number of perfect matchings, where the best known deterministic algorithms take exponential time. For other problems such as primality testing and univariate polynomial identity testing, randomized algorithms offer a polynomial speedup over the best known deterministic ones. Finally, property testing has problems that admit incredibly efficient randomized algorithms [GGR98], in fact sublinear, but provably require linear time deterministically. Informally, randomization owes its success to the prevalence of a seemingly paradoxical phenomenon: most courses of action for an algorithm may be good, yet it might be hard to pinpoint one good course of action.

On the other hand, there are upper bounds on the power of randomization. Assuming plausible circuit lower bounds, any randomized algorithm running in time t on inputs of length n can be simulated deterministically by an algorithm running in time polynomial in t and n [NW94, IW97].[1] In other words, under plausible assumptions, $\mathbf{BPP} = \mathbf{P}$. Concretely[2]:

Theorem 1.1 (previously known derandomization [NW94, IW97]). *For every positive integer n and a constant $\alpha < 1$, assume there exists a function $f \in \mathbf{DTIME}(\widetilde{O}(2^n))$ that requires circuits of size $2^{(1-\alpha)n}$. Let A be a bounded-error probabilistic algorithm, accepting a language[3] L, that on inputs of length n runs in time $t = t(n)$. Then, there exists a deterministic algorithm that accepts L and runs in time $\mathrm{poly}(t, n)$.*

The runtime of the deterministic algorithm of Theorem 1.1 is a large polynomial, for reasons inherent to the proof technique of Theorem 1.1, as we will explain shortly. There are other proofs of Theorem 1.1 [SU05, Uma03] and they too yield a large polynomial running time. Our main theorem gives a derandomization with running time $t \cdot \max\{t, n\}^{1+\alpha}$ for arbitrarily small constant $\alpha > 0$. It relies on a somewhat stronger assumption than that of Theorem 1.1, namely ruling out certain *nondeterministic* circuits for f.

Theorem 1.2 (general derandomization, see Theorem 9.4). *For every positive integer n and a small enough constant $\alpha > 0$, assume there exists a function $f \in \mathbf{DTIME}(\widetilde{O}(2^n))$ that requires circuits with $\mathbf{FNP}$ gates of size $2^{(1-\alpha)n}$. Let A be a bounded-error probabilistic algorithm, accepting a language L, that on inputs of length n runs in time $t = t(n)$. Then, there exists a deterministic algorithm that accepts L and runs in time $t \cdot \max\{t, n\}^{1+O(\alpha)}$.*

Our derandomized running time is nearly tight for some examples above, such as those from property testing (at least for black-box derandomization). To further illustrate the tightness, consider the problem of *univariate polynomial identity testing*, in which we test the identity of two arithmetic circuits with one variable, degree n, and $O(n)$ wires, over a field of cardinality $\mathrm{poly}(n)$. Univariate polynomial identity testing is solvable in $\widetilde{O}(n)$ randomized time and $\widetilde{O}(n^2)$ deterministic time. Williams [Wil16] showed that coming up with an $n^{1.999}$-time algorithm, even a nondeterministic

[1]On first reading, it may be helpful to just consider the case $t \geq n$.

[2]We rephrase from the usual statement, to better compare with our new result.

[3]Here, and throughout the paper, languages (decision problems) can be replaced by other problems, e.g., promise problems or functions with non-binary output. Also, note that we consider algorithms in the Turing machine model. Running times in other models, such as the RAM model, may differ, but one can change the hardness assumptions accordingly.

one, would refute NSETH[4]. Interestingly, the problem of *multivariate* identity testing also admits a randomized $\widetilde{O}(n)$-time algorithm. Thus, under the complexity-theoretic assumption of Theorem 1.2, and assuming NSETH holds, the time complexity of *both* univariate and multivariate identity testing is settled at roughly quadratic.

While our hardness assumption is not ideal, related and sometimes stronger complexity-theoretic assumptions were used before to derandomize both deterministic and nondeterministic classes, as well as to construct various pseudorandomness primitives. Specifically, hardness against single-valued nondeterministic (SVN) circuits[5] was used in [SU05, MV05], against nondeterministic circuits in [AK97, MV05, SU06, SU07, BOV07, Dru13], against circuits with **NP** gates in [KvM02, GW02] and even against circuits with **PH** gates in [TV00, AIKS16, AASY16, AS17].

Towards proving our main theorem, we also derandomize algorithms that err rarely, known as *quantified derandomization*. First studied by Goldreich and Wigderson [GW14], these are algorithms that err only on a small *number* of the possible randomness strings (their error *probability* is extremely small). Our derandomization of such algorithms produces deterministic algorithms whose running time is *similar* to the runtime of the randomized algorithm, up to a quadratic preprocessing step[6].

Theorem 1.3 (quantified derandomization, see Corollary 5.3)**.** *There exists a constant $c \geq 1$ such that for every positive integer n and a small enough constant α the following holds. Assume there exists a function $f \in \mathbf{DTIME}(\widetilde{O}(2^n))$ that requires SVN circuits of size $2^{(1-\alpha)n}$. Let A be a bounded-error probabilistic algorithm, accepting a language L, that on inputs of length n runs in time $t = t(n)$ and for every input, errs on at most $2^{t^{1-c\alpha}}$ randomness strings.*

Then, there exists a deterministic algorithm that accepts L and runs in time $t \cdot \max\{t, n\}^{O(\alpha)} + t_P$, where the $t_P = t^{2+O(\alpha)}$ term corresponds to a step that can be precomputed for all algorithms with running time t.

Unconditional quantified derandomization has been successful for restricted classes of computation, and also sufficiently-good quantified derandomization has been shown to imply circuit lower bounds [GW14, Tel18, Tel19, CT19].

Like Theorem 1.1, we take the black-box approach for derandomization and prove Theorem 1.2 by constructing a *pseudorandom generator*. A pseudorandom generator (PRG) with error ε is a function that maps a short seed to a t-bit string that is indistinguishable from uniform random bits by any time t algorithm, up to error ε. We can derandomize an algorithm using a PRG by enumerating over all possible seeds, and running the randomized algorithm on the output of the generator on each seed. The number of possible seeds determines the slowdown[7] of the deterministic algorithm, and this number was $\text{poly}(t, n)$ for a large polynomial prior to this work.

Non-explicitly, there exists a pseudorandom generator against all algorithms running in time t on inputs of length n that uses only $O(\max\{t, n\})$ seeds, but it is not necessarily efficiently computable. In this work we construct an explicit PRG with only $\max\{t, n\}^{1+O(\alpha)}$ seeds.

Theorem 1.4 (pseudorandom generator, see Theorem 9.1)**.** *For every positive integer n and constants $0 < \varepsilon, \alpha < 1$, assume there exists a function $f \in \mathbf{E}$ that requires circuits with **FNP** gates of size $2^{(1-\alpha)n}$.*

[4]The Nondeterministic Strong Exponential-Time Hypothesis asserts that refuting unsatisfiable k-CNFs requires nondeterministic $2^{n-o(n)}$ time for unbounded k.

[5]SVN circuits can be seen as the nonuniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$. See Definition 2.1.

[6]The preprocessing step involves encoding the truth table of f. Computing the truth table may take quadratic time, but can be made efficient if f admits fast batch evaluation.

[7]We say a derandomization has slowdown of $S = S(n)$ if $t_{\mathsf{det}}/t_{\mathsf{rand}} = S$, where $t_{\mathsf{det}} = t_{\mathsf{det}}(n)$ and $t_{\mathsf{rand}} = t_{\mathsf{rand}}(n)$ are the running times of the relevant deterministic and randomized algorithms on inputs of length n.

Then, there exists an explicit PRG

$$\overline{G}^f \colon \{0,1\}^{(1+O(\alpha))\log s} \to \{0,1\}^s$$

with error ε, fooling circuits of size $s = n^{1-O(\alpha)}$.

We consider f on roughly $\log s$ bits of input, so the truth table of f consists of roughly s bits. The pseudorandom generator converts those s bits of (worst-case) hardness into s bits of pseudorandomness. Assuming $f \in \mathbf{DTIME}(\widetilde{O}(2^n))$, our PRG is computable in time $\max\{t,n\}^{2+O(\alpha)}$.

Let us compare the parameters of Theorem 1.4 to the prior state-of-the-art PRG given by Umans [Uma03]. There, the seed is of length $c_{\mathsf{U}} \log s$ for a large constant $c_{\mathsf{U}} > 1$, and $s = n^{\gamma_{\mathsf{U}}}$ for a small constant $\gamma_{\mathsf{U}} < 1$. Consequently, derandomization using Uman's PRG would incur a slowdown of at least $s^{c_{\mathsf{U}}/\gamma_{\mathsf{U}}}$ in the running time, whereas in this paper we bring this factor down to $s^{1+O(\alpha)}$. Roughly speaking, we manage to transform almost all the hardness to pseudorandom bits ($s = n^{1-O(\alpha)}$), and we manage to do so using a short seed. The downside of Theorem 1.4 compared to previous works is that we assume f is hard for a seemingly stronger class of circuits.

An important milestone in constructing our PRG $\overline{G}^f$ is the construction of a *pseudoentropy generator* (PEG) with an especially small seed, from which Theorem 1.3, our quantified derandomization, follows. In a PEG, the output distribution is computationally-indistinguishable, up to some error, from *some* high min-entropy distribution[8].

Theorem 1.5 (pseudoentropy generator, see Theorem 4.2)**.** *For every positive integer n and constants $0 < \varepsilon < 1$, $0 < \alpha < \frac{1}{8}$, assume there exists a function $f \in \mathbf{E}$ that requires SVN circuits of size $2^{(1-\alpha)n}$. Then, there exists an explicit PEG*

$$G^f \colon \{0,1\}^d \to \{0,1\}^s$$

with error ε, $d = 6\alpha \log n$ and $s = n^{1-4\alpha}$, outputting pseudoentropy $k = n^{1-8\alpha}$ fooling circuits of size s.

For simplicity, we phrase and prove Theorem 1.4 and Theorem 1.5 for a constant error ε, however we can also handle slightly sub-constant error (in particular, any $\varepsilon = 1/\operatorname{polylog} n$).

We note that pseudoentropy generators are analogous to randomness *condensers* in roughly the sense that pseudorandom generators are analogous to randomness *extractors*, where the hard function plays the role of a high min-entropy source [Tre01, TZ04].

Finally, our complexity-theoretic assumptions allow us to easily adapt existing techniques for derandomizing $\mathbf{AM}$ and use our PRG from Theorem 1.4 in order to put $\mathbf{AM}$ in $\mathbf{NP}$ with a smaller slowdown. See Section 9.1 for the details.

In the remainder of the introduction we describe the ideas behind the proofs of Theorem 1.4 and Theorem 1.5.

1.2 Beating the Hybrid Argument

Existing proofs of Theorem 1.1 can be viewed as first constructing a pseudorandom generator that extends its seed by a single bit, and then converting it into a pseudorandom generator that outputs t bits. Interestingly, Sudan, Trevisan, and Vadhan [STV01] constructed PRGs with small seed as in Theorem 1.4 for the single bit case. They did this by using binary locally decodable codes to encode the truth table of f. The pseudorandom generator outputs a random bit of the encoding, and hence the number of seeds corresponds to the length of the encoding. Since there are locally decodable

[8]Indeed, our notion of computational entropy allows the high min-entropy distribution to depend on the distinguishing algorithm. See Section 2.5 for the precise details.

codes of linear length and a sub-linear number of queries (e.g., codes obtained from Reed-Muller composed with Hadamard, or tensor codes [Yek12]), there are single bit PRGs with a small number of seeds.

The large loss in time in Theorem 1.1 originates from the extension of a single bit output to t bits of output. The loss has several manifestations in each of the existing proofs of Theorem 1.1. The use of combinatorial designs in [NW94, IW97] inherently doubles the length of the seed. The use of the Reed-Muller code in [SU05, Uma03] inherently limits the length of the generator's output. Moreover, the analyses of these constructions go through the infamously-hard-to-beat *hybrid argument* [FSUV13], which incurs an additional multiplicative factor of at least t to the number of seeds: For t bits to be indistinguishable from uniform, it must be that each bit is unpredictable given the previous bits, and the prediction errors add up across the t bits. Hence, each prediction error has to be smaller than $\frac{1}{t}$. However, to produce a single bit that is $\frac{1}{t}$-close to uniform one has to multiply the number of seeds by a factor of at least t.[9]

In order to prove Theorem 1.4 we must beat the hybrid argument. We do that by first constructing a pseudoentropy generator as in Theorem 1.5, later to be transformed to a pseudorandom generator. Since we no longer require uniform-looking bits, we no longer need an error of $\frac{1}{t}$ per bit. Instead, we output a large number of (imperfect) bits with constant error, which can evidently be done with a small number of seeds. Indeed, Barak, Shaltiel and Wigderson [BSW03] suggested that paradigm, of achieving pseudoentropy as a stepping stone towards pseudorandomness as a way to bypass the weakness of the hybrid argument, and here we fulfill this vision.

1.3 Locally Decodable Codes All the Way Down

As explained above, it was known that single bit pseudorandom generators follow from binary locally decodable codes [STV01]. We show that *pseudoentropy* generators that output many bits follow from locally decodable codes over a large alphabet.

More accurately, we consider locally *list recoverable* codes. In a list recoverable code $\mathcal{C} \subseteq \Sigma^n$ for agreement ε, we are given oracle access to lists $S_1, \ldots, S_n \subseteq \Sigma$ for which $\sum_{i=1}^{n} |S_i| \leq \ell$, and we are guaranteed that there are at most L codewords $c \in \mathcal{C}$ satisfying $c_i \in S_i$ for at least ε-fraction of the i-s. We say $\mathcal{C}$ admits *local* list recovery if there exist circuits $A_1, \ldots, A_L$ with oracle access to the lists $S_1, \ldots, S_n$, each A_i having at most Q oracle gates, such that for every codeword $c = \mathcal{C}(x)$ satisfying $c_i \in S_i$ for at least ε-fraction of the i-s, there exists $j \in [L]$ such that $x = A_j(\cdot)$.[10] Initially, list recoverable codes were used as an intermediate step for constructing list decodable codes (e.g., in [GI01, GI02, GI03]) but have since gained independent interest, with several applications and dedicated constructions (e.g., [HIOS15, HRZW17, HRZW17, HW18, RW18]). Moreover, many of the recent list decoding algorithms are in fact algorithms for list recovery.

We prove that locally list recoverable codes give rise to pseudoentropy generators.

Theorem 1.6 (PEGs from locally list recoverable codes, see Theorem 4.1)**.** *Assume $f \in \{0,1\}^t$ is a truth table of a function requiring SVN circuits of size $t^{1-\alpha}$, and let $\alpha' > \alpha$ be any constant.*

Let $\mathcal{C} \colon \{0,1\}^t \to \Sigma^m$ be a locally list recoverable code for agreement $\varepsilon > 0$ and input lists size ℓ, for which each decoding circuit has size $s_{\mathcal{C}} \leq t^{1-\alpha'}$ and makes at most Q oracle queries to the lists. Then $G^f \colon [m] \to \Sigma$, defined by

$$G^f(z) = \mathcal{C}(f)_z,$$

[9] The loss due to the hybrid argument is an inherent artifact when analyzing any *reconstructive* PRG, see e.g. [Sha04].

[10] Unlike in standard literature, where algorithmic aspects are crucial, we do not require (and will not achieve) uniform generation of the A_i-s. Also, each query we make to a list S_i gives us a *single* element of S_i, and we do not get to iterate over the entire list. For the exact definition, refer to Section 2.2.

is a PEG with error ε, outputting $k = \frac{\log \ell}{2}$ pseudoentropy fooling circuits of size $\frac{t^{1-\alpha'}}{Q}$.

Note that in order to get a good PEG, the alphabet Σ of the list recoverable code $\mathcal{C}$ must be large, and the lists size ℓ must be large as well. In fact, they both need to be *exponential* in t, while the decoder should run in time smaller than t. It is atypical to require such a large ℓ, and in standard literature, where the decoder typically iterates over the lists, one requires $\ell \ll m$ in order to get a satisfactory bound on L. We will soon briefly discuss the construction of $\mathcal{C}$ and how we facilitate efficient access to the lists.

A few words about our notion of pseudoentropy are in order. We say a random variable $X \sim \{0,1\}^n$ has k pseudoentropy fooling circuits of size s, up to ε error, if for every circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size s there exists $Y \sim \{0,1\}^n$ with min-entropy k such that $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \le \varepsilon$.[11] This definition, coined *metric pseudoentropy*, was first studied by Barak et al. [BSW03] and later in various works in cryptography [DP08, CKLR11, FR12, Wic13, Sko15, FOR15, SGP15]. Metric pseudoentropy differs from the widely-used notion of HILL pseudoentropy [HILL99] where the high min-entropy random variable does not depend on the distinguishing circuit. Although seemingly weaker, this definition allows us to derandomize algorithms that err rarely, as random variables with high metric pseudoentropy cannot have large weight on small sets (see Section 5 for the details).

We prove Theorem 1.6 by drawing a connection between locally list recoverable codes and compression-based pseudoentropy, or Yao pseudoentropy [Yao82]. We then utilize the fact that one can get metric pseudoentropy from Yao pseudoentropy by allowing the distinguisher D to be an SVN circuit (see Section 2.5.1). Briefly, we first show that if $G^f(U)$, i.e. a random coordinate of $\mathcal{C}(f)$, does not have sufficiently high Yao pseudoentropy, then we can replace each of the Q queries to the lists $S_1, \dots, S_m$ in the decoding circuits $A_1, \dots, A_L$ of $\mathcal{C}$ by queries to *small* circuits that encode the lists. We then follow [BSW03] and use a hash function to map elements in (the large) Σ into a universe of size roughly ℓ. We use nonuniformity to point to the elements in the size-ℓ universe, and nondeterminism to guess the pre-image of the hash function. Setting the parameters accordingly, we eventually get a small nondeterministic circuit computing f, thus contradicting its hardness. The rest of the details are given in Section 4.

It is interesting to draw an analogy between Theorem 1.6 and its information-theoretic counterparts (which we formally define in Section 2.4). The work of Ta-Shma and Zuckerman [TZ04], following Trevisan [Tre01], shows the equivalence between extractors with multiple output bits and *soft-decision decoding*, which we will not define here. Roughly speaking, if such codes are equipped with an efficient local decoding procedure, they give rise to PRGs. Soft-decision decoding generalizes list recoverable codes, and so every extractor can be used to construct a list recoverable code with suitable parameters. We argue that this result *interpolates* for lower min-entropies too. As already observed in [GUV09] under a somewhat different terminology, every list recoverable code for agreement ε gives rise to a condenser condensing $k = \log \frac{L}{\varepsilon}$ min-entropy to $k' = \log \frac{\ell}{m}$ min-entropy with error $O(\varepsilon)$, and each such $k \to k'$ condenser with error ε implies a list recoverable code for agreement $O(\varepsilon)$ with $\ell = O(\varepsilon m 2^{k'})$ and $L = 2^k$. For the formal statement and its proof, see Appendix A. Thus, in a way, Theorem 1.6 is a computational manifestation of these connections.

[11] A random variable $Y \sim \{0,1\}^n$ has min-entropy k if for every $y \in \mathrm{Supp}(Y)$, $\Pr[Y = y] \le 2^{-k}$.

1.3.1 Constructing the List Recoverable Code $\mathcal{C}$

We construct our locally list recoverable code $\mathcal{C}\colon \{0,1\}^t \to \Sigma^m$ over a large alphabet from locally decodable codes over a small alphabet, as follows. Let

$$\mathcal{C}_{\mathsf{LDC}}\colon \{0,1\}^t \to \mathbb{F}_q^{\bar{t}}$$

be some locally decodable code having relatively high rate. Next, we pick a biregular sampler

$$\Gamma\colon [\bar{t}] \times [D] \to [m]$$

with right-degree a which is close to t. Given $f \in \{0,1\}^t$, each coordinate of $\mathcal{C}(f)$ corresponds to a large pseudorandom subset of $\mathcal{C}_{\mathsf{LDC}}(f)$ determined by Γ. Namely, for every $z \in [m]$,

$$\mathcal{C}(f)_z = \mathcal{C}_{\mathsf{LDC}}(f)_{\Gamma^{-1}(z)} = \mathcal{C}_{\mathsf{LDC}}(f)_{\Gamma^{-1}(z,1)} \circ \ldots \circ \mathcal{C}_{\mathsf{LDC}}(f)_{\Gamma^{-1}(z,a)}.$$

Here $\Gamma^{-1}(z,i)$ denotes the i-th neighbor of z. Clearly, $\mathcal{C}$ is defined over a large alphabet. Also, we have good samplers so m can be very small, roughly t^{α}, which in turn implies that the seed length of our PEG G^f is very small. But why does this construction admit a local list recovery procedure for exponential-sized lists?

Consider the task of decoding some entry $x \in [t]$ of f given lists $S_1, \ldots, S_m$, where we are guaranteed that $\mathcal{C}_f(z) \in S_z$ for at least ε-fraction of the z-s. Recall that $\mathcal{C}_{\mathsf{LDC}}$ is itself equipped with a local list decoding procedure, so in particular there exists a randomized circuit A_{LDC} that locally decodes f from a noisy version of it. How do we mimic a good enough noisy version of f?

- Run A_{LDC} on the input x.
- Whenever A_{LDC} wishes to query a certain coordinate of $\mathcal{C}_{\mathsf{LDC}}(f)$, say coordinate $i \in [\bar{t}]$, we choose a random neighbor of i in Γ, say $j \in [m]$.
- We now want to query the list S_j for the correct value of $\mathcal{C}_{\mathsf{LDC}}(f)_i$, and we use a *hardwired advice* to pinpoint the specific entry in S_j to be read. From the list's entry we can deduce a guess for $\mathcal{C}_{\mathsf{LDC}}(f)_i$, i.e., a coordinate in the noisy version of f.
- Recall that there are also quite a few bad lists. Using the sampling property, we can guarantee that with high probability, a sufficient number of queries will be made to good lists.

For the complete details, see Section 3. We stress that nonuniformity plays a crucial role here, throughout the analysis. Also, since the lists $S_1, \ldots, S_m$ are fixed (and in fact, determined by G^f), and $Q \approx m \ll t$, we are allowed to *fix* the advice describing the pointers to all lists, each pointer is of length roughly $\log \ell$, and we do not have to worry about the possibly different query indices for different x-s and different randomness strings. As our only bound on ℓ stems from the bound on the decoding circuit size, $m \log \ell \approx t$ and so ℓ can be exponentially large.

We note that using expander- (or sampler-) based transformations in coding theory is quite common, and can be found in several other works, e.g., in [ABN+92, AEL95, GI02, GI03, KMRZS17, DHK+19].

1.4 From Pseudoentropy to Pseudorandom Bits

Finally, we construct our pseudorandom generator $\overline{G}^f$ from Theorem 1.4 by composing the pseudoentropy generator G^f of Theorem 1.5 with a new construction of an extractor with seed length close to $\log n$ (for constant error), supporting $n^{1-\alpha}$ min-entropy for any $\alpha < \frac{1}{2}$.

Theorem 1.7 (short seed extractor, see Theorem 8.4)**.** *There exists a constant $c \geq 1$ such that the following holds for any constant $\alpha < \frac{1}{2}$. For every positive integer n and every $\varepsilon \geq cn^{-\frac{1}{2}+\alpha}$, there exists an explicit extractor $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with error ε, supporting min-entropy $k \geq n^{1-\alpha}$, where $d = (1+c\alpha)\log n + c\log\frac{1}{\varepsilon}$ and $m = \frac{1}{c}n^{1-2\alpha}$.*

This adds to the short list of extractors with almost the right dependence on n, currently including [TZS06, Zuc07] and one-bit extractors coming from good list decodable codes. Our construction essentially follows a construction given in [TZS06]; however, there it was analyzed for smaller min-entropies. The construction and its analysis are given in Section 8.

The seed of the pseudorandom generator consists of the (very short) seed of the pseudoentropy generator, as well as the seed of the extractor, which still gives us $(1 + O(\alpha))\log t$. It is not immediately clear why such a composition works, and indeed we make quite a few modifications for our argument to go through, which we will discuss shortly.

Theorem 1.8 (composition, see Theorem 6.9)**.** *Assume there exists a function in $\mathbf{E}$ that requires exponential-size circuits and suppose $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is an extractor with error ε supporting min-entropy k, computable by a circuit of size s_{Ext}.*

Let $X \sim \{0,1\}^n$ be a random variable with pseudoentropy k fooling circuits of size $s' = O(s + s_{\mathsf{Ext}})$, up to ε error, where we allow the circuits to output real values in $[0,1]$ and have oracle gates to some fixed function problem in $\mathbf{FNP}$. Then, $\mathsf{Ext}(X, U_d)$ fools deterministic circuits of size s, up to error $O(\varepsilon)$.

Note that we require more from the circuit class that we need X, the output distribution of our PEG, to fool. Instead of fooling every small SVN circuit, we need X to have high pseudoentropy against *real-valued* small circuits having **FNP** gates.

Previously, composition was known to work when the output of the PEG was indistinguishable from a *universal* distribution with high min entropy. i.e., had high HILL pseudoentropy [BSW03].[12] However, the definition of PEGs that we use allows for the high min-entropy random variable to depend on the distinguisher, and the natural distinguisher for the composed construction invokes the extractor. In this case, the high min-entropy random variable may depend on a specific seed of the extractor, and some of the seeds of the extractors are destined to fail!

The above obstacle is where our stronger class of distinguishers enters the picture. Barak et al. [BSW03] showed that one can move from HILL pseudoentropy to metric pseudoentropy for real-valued distinguishers. Unfortunately, their argument suffers from a considerable loss in circuits size which we cannot afford, and seems inherent. The idea to overcome this loss is to use derandomized sampling to approximate some averaging of the extractor on all possible seeds. We defer the rest of the details to Section 6.

The fact that we need a stronger pseudoentropy guarantee affects our paradigm of PEGs from locally list recoverable codes. For example, we are no longer guaranteed access to *small lists* $S_1, \ldots, S_m \subseteq \Sigma$ for which $\mathcal{C}(f)_z \in S_z$ for at least ε of the z-s. Instead, we ought to handle weight functions $S_1, \ldots, S_m \colon \Sigma \to [0,1]$ having *small expectation* satisfying $\mathbb{E}[S_z(\mathcal{C}(f)_z)] \geq \varepsilon$, which puts us in the soft-decision decoding regime. Luckily, the same PEG G^f still works, and only its analysis needs to be modified. This is the content of Section 7. On the way, we extend the notion of compression-based pseudoentropy to real-valued distinguishers (see Section 6.3).

1.5 Open Problems

Some challenges remain to be tackled. We list only a few of them.

[12] A notable example is [STV01], where they show that the Nisan-Wigderson PRG [NW94] outputs pseudoentropy when f is only mildly hard.

1. Can we achieve a nearly linear time quantified derandomization without any preprocessing?
2. Can we achieve similar results assuming f is exponentially-hard for *deterministic* – rather than nondeterministic – circuits?
3. Can we construct a pseudoentropy generator with short seed outputting pseudoentropy $s^{1-o(1)}$ or $\Omega(s)$? This would allow us to very efficiently derandomize algorithms that err less rarely.
4. Can we achieve lower error $1/n^{o(1)} \leq \varepsilon \ll 1/\operatorname{polylog} n$ for the pseudorandom generator, while still maintaining a short seed length?
5. Can we derandomize any algorithm that runs in time t on inputs of length n in time close to tn, as opposed to $t \cdot \max\{t, n\}$? This would match the runtime of Adleman's *nonuniform* derandomization [Adl78]. Note that *for every fixed algorithm* on input size n, independent of its runtime, there exists a pseudorandom generator against this algorithm that has only $O(n)$ seeds.

2 Preliminaries

The density of a set $B \subseteq A$ is $\mu_A(B) = \frac{|B|}{|A|}$ (when A is clear from context, we shall omit it). For a positive integer A, we denote by $[A]$ the set $\{1, \ldots, A\}$. For a set A, by $x \sim A$ we mean x is drawn uniformly at random from the uniform distribution over the elements of A.

For a function $f \colon \Omega_1 \to \Omega_2$, we say f is *explicit* if there exists a deterministic procedure that runs in time $\operatorname{poly}(\log |\Omega_1|)$ and computes f. If Ω_2 the contains 0, we denote by $\operatorname{Supp}(f)$ the set $\{x \in \Omega_1 : f(x) \neq 0\}$. Finally, for $f, g \colon \mathbb{N} \to \mathbb{N}$, we say that $f(n) = \widetilde{O}(g(n))$ if there exists a constant k such that $f(n) = O(g(n) \log^k g(n))$. All the logarithms are in base 2.

2.1 Circuits, Nondeterministic Circuits and Worst-Case Hardness

The size of a Boolean circuit is the number of its *wires*. We will extensively use the fact that $\mathbf{DTIME}(t(n)) \subseteq \mathbf{SIZE}(O(t(n) \log t(n)))$, where $\mathbf{SIZE}(t(n))$ is the set of languages $L \subseteq \{0,1\}^n$ for which there exists a circuit family $\{C_n\}_n$, such that each C_n is of size $t(n)$, and for every $x \in \{0,1\}^n$ it holds that $x \in L$ if and only if $C_n(x) = 1$ [PF79].

Definition 2.1 (SVN circuit, [SU05])**.** *A single-valued nondeterministic (SVN) circuit $C \colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ is a circuit that gets as inputs an input string $x \in \{0,1\}^n$ and a witness string $y \in \{0,1\}^w$ and outputs $C(x,y)$ with an additional flag $C(x,y)_{\mathsf{check}} \in \{0,1\}$. We say that C computes $f \colon \{0,1\}^n \to \{0,1\}^m$ if the following holds.*

1. *For every $x \in \{0,1\}^n$ there exists $y \in \{0,1\}^w$ such that $C(x,y)_{\mathsf{check}} = 1$.*
2. *For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^w$ such that $C(x,y)_{\mathsf{check}} = 1$ it holds that $C(x,y) = f(x)$.*

Definition 2.2 (hardness against SVN circuits)**.** *We let $\mathsf{size}_{SV}(f)$ to denote the smallest SVN circuit that computes f. We say $f \colon \{0,1\}^n \to \{0,1\}$ requires exponential-size SVN circuits if $\mathsf{size}_{SV}(f) > 2^{\delta n}$ for some constant $\delta > 0$.*

SVN circuits can be viewed as the non-uniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$. Note that if $f \colon \{0,1\}^n \to \{0,1\}$ is computed by an SVN circuit then $\neg f$ is computable by an SVN circuit of the same size.

We also give the definition for nondeterministic (and co-nondeterministic) circuits outputting one bit.

Definition 2.3. *A* nondeterministic circuit C *gets as input an input string* $x \in \{0,1\}^n$ *and a witness string* $y \in \{0,1\}^m$*, and we say* C *computes* $f\colon \{0,1\}^n \to \{0,1\}$ *if it holds that* $f(x) = 1$ *iff there exists* y *such that* $C(x, y) = 1$*. For* f *to be computed by a* co-nondeterministic circuit C*, we require that* $f(x) = 1$ *iff for every* y*,* $C(x, y) = 1$*.*

As shown in [SU05], if $f\colon \{0,1\}^n \to \{0,1\}$ is such that $\mathsf{size}_{SV}(f) > s$ then there exists a function $f\colon \{0,1\}^{n+1} \to \{0,1\}$ which cannot be computable by either nondeterministic circuits or co-nondeterministic circuits of size $O(s)$. We will also work with a stronger model of computation, in which we allow oracle gates to fixed function problems in **FNP**. For part of our work we only need oracle gates to fixed problems in **FP**.

Definition 2.4 (**FP**-circuit, **FNP**-circuit)**.** *A* **FP**-circuit *(or* **FNP**-circuit*) is a Boolean circuit which can have gates for some fixed function problem in* **FP** *(or* **FNP***, respectively) in addition to the usual* $\wedge$*,* $\vee$ *and* $\neg$ *gates.*

Definition 2.5 (hardness against **FNP**-circuits)**.** *We let* $\mathsf{size}_{\mathbf{FNP}}(f)$ *denote the smallest* **FNP***-circuit that computes* f*. We say* $f\colon \{0,1\}^n \to \{0,1\}$ *requires exponential-size* **FNP***-circuits if* $\mathsf{size}_{\mathbf{FNP}}(f) > 2^{(1-\alpha)n}$ *for some constant* $0 < \alpha < 1$*.*

We will also consider circuits (and **FNP**-circuits) that output real values in $[0, 1]$. A circuit $D\colon \{0,1\}^n \to [0,1]$ is a Boolean circuit outputting multiple bits which are the binary representation of the desired real value. Note that there is a technical difficulty in the fact that arbitrary real numbers require an arbitrarily large number of bits to represent. However, for our purposes it suffices to consider circuits D with a small number of output bits. Specifically, if a circuit D outputs an arbitrary real value ε, we can instead consider the circuit that rounds ε to $\min_{i\in\mathbb{N}} |\varepsilon - i \cdot 2^{-\ell}|$, which can be encoded using ℓ bits. Doing so incurs an error of at most $2^{-\ell}$, which is acceptable for our purposes. In fact any $\ell = \omega(1)$ will be satisfactory, as our error guarantees will be constant.

2.2 Error Correcting Codes

A fundamental primitive in complexity theory is the error correcting code.

Definition 2.6 (relative distance)**.** *For some alphabet* Σ*, let* $x, y \in \Sigma^n$*. The relative distance of* x, y*, denoted* $\delta(x, y)$*, is the fraction of coordinates on which* x, y *differ. That is,* $\delta(x, y) = \Pr_{i\sim[n]}[x_i \neq y_i]$*.*

Definition 2.7 (linear code)**.** *Let* n, k, q *be positive integers with* q *a prime power and* $0 \leq \delta \leq 1$*. We say that* $\mathcal{C}\colon \mathbb{F}_q^k \to \mathbb{F}_q^n$ *is an* $[n, k, \delta]_q$ *code if* $\mathcal{C}$ *is a linear transformation and for all* $x, y \in \Sigma^k$*,* $\delta(\mathcal{C}(x), \mathcal{C}(y)) \geq \delta$*. By abuse of notation, we often use* $\mathcal{C}$ *to denote* $\mathrm{Im}(\mathcal{C}) \subseteq \Sigma^n$*.*

Definition 2.8 (binary list-decodable codes)**.** *A code* $\mathcal{C} \subseteq \Sigma^n$ *is* (τ, L) *list decodable if for every* $w \in \Sigma^n$*,* $|\{c \in \mathcal{C} : \delta(w, c) \leq \tau\}| \leq L$*.*

In our pseudoentropy generator construction, we make use of locally list decodable codes, and their generalization – list recoverable codes.

Definition 2.9 (locally list decodable code, implicit model)**.** *Let* $\mathcal{C}\colon \Sigma^k \to \Sigma^n$*. Let* Q, L, s *be positive integers and* $0 < \varepsilon, \zeta < 1$*. We say that* $\mathcal{C}$ *is* $(Q, \varepsilon, \zeta, s, L)$ *locally list decodable if there exist randomized circuits* $A_1, \ldots, A_L$*, each of size* s *that satisfy the following.*

- *Each A_j has oracle access to a received word $r \in \Sigma^n$, and makes at most Q oracle queries to coordinates in r.*
- *For every codeword $c = \mathcal{C}(x)$ such that c agrees with r in at least ε places, there exists $j \in [L]$ such that for every $i \in [k]$, we have $A_j(i) \neq x_i$ with probability at most ζ over the randomness used by A_j.*

For convenience, we often say in words that $\mathcal{C}$ is an LLDC using Q queries, handling agreement of at least ε, with failure probability ζ, circuit size s, and output list size L.

Definition 2.10 (list recoverable code)**.** *Let $\mathcal{C} \colon \Sigma^k \to \Sigma^n$. For positive integers ℓ, L and $\varepsilon > 0$ we say that $\mathcal{C}$ is (ε, ℓ, L) list recoverable if the following holds. For every sequence of sets $S_1, \ldots, S_n \subseteq \Sigma$ so that $\sum_{i=1}^{n} |S_i| \leq \ell$ there are at most L codewords $c \in \mathcal{C}$ satisfying $c_i \in S_i$ for at least ε fraction of the i-s.*

We present a notion of list recoverable codes convenient for our purposes.

Definition 2.11 (locally list recoverable code, implicit model)**.** *Let $\mathcal{C} \colon \Sigma^k \to \Sigma^n$. For positive integers ℓ, L, Q, and $0 < \varepsilon < 1$ we say that $\mathcal{C}$ is $(Q, \varepsilon, \ell, s)$ list recoverable if there exist circuits $A_1, \ldots, A_L$, each of size s, that satisfy the following.*

- *Each A_j takes as input $i \in [k]$ and also has oracle access to $\bar{S} = (S_1, \ldots, S_n)$, where each $S_z \subseteq \Sigma$ and $\sum_{z=1}^{n} |S_z| \leq \ell$. A_j uses at most Q oracle gates, and each oracle query constitutes a* ***single*** *probe to a specific set. That is, each oracle gate takes as input a string $y \in \{0,1\}^{\log \ell}$ and outputs a symbol in Σ. Note here that we think of having a* ***single*** *oracle for $\bar{S}$, where the input string to the oracle specifies both the list and the index in the list.*
- *For every codeword $c = \mathcal{C}(x)$ satisfying $c_z \in S_z$ for at least ε fraction of the i-s, there exists $j \in [L]$ such that for every $i \in [k]$, we have $A_j(i) = x_i$.*

For convenience, we often say in words that $\mathcal{C}$ is a LLRC using Q queries, handling agreement at least ε, with circuit size s and input list size ℓ and output list size L.

Note that under the above definition of list recovery, we do not insist on a *uniform* generation of the decoding circuits, and that unlike in our LLDC definition, the circuits are now deterministic. We mention that in standard literature on locally list recoverable codes, the decoder is randomized (and thus has some small probability of failure), and each query to the oracle is of the form "what is the entire list of symbols for the z-th coordinate." We deviate from the standard notion in our work because we wish to efficiently handle lists of exponential size, and exhibit a **deterministic** recovering circuit that contradicts our hardness assumption. Finally, the output list size L, which is usually a notable parameter in this line of research, is only implicit here (i.e., the size of each decoding circuit is bound to have size roughly $\Omega(\log L)$).

2.3 Random Variables, Min-Entropy

The *support* of a random variable X distributed over some domain Ω is the set of $x \in \Omega$ for which $\Pr[X = x] \neq 0$, which we denote by $\mathrm{Supp}(X)$.

The *statistical distance* between two random variables X and Y on the same domain Ω is defined as $|X - Y| = \max_{A \subseteq \Omega}(\Pr[X \in A] - \Pr[Y \in A])$. If $|X - Y| \leq \varepsilon$ we say ε-close to Y and denote it by $X \approx_{\varepsilon} Y$. We denote by U_n the random variable distributed uniformly over $\{0,1\}^n$. We say a random variable is *flat* if it is uniform over its support.

For a function $f \colon \Omega_1 \to \Omega_2$ and a random variable X distributed over Ω_1, $f(X)$ is the random variable distributed over Ω_2 obtained by choosing x according to X and computing $f(x)$. For a set $A \subseteq \Omega_1$, $f(A) = \{f(x) : x \in A\}$. For every $f\colon \Omega_1 \to \Omega_2$ and two random variables X and Y distributed over Ω_1 it holds that $|f(X) - f(Y)| \leq |X - Y|$.

The *min-entropy* of a random variable X is defined by

$$H_\infty(X) = \min_{x \in \mathrm{Supp}(X)} \log \frac{1}{\Pr[X = x]}$$

For some $\varepsilon > 0$, we define the *smooth min-entropy* of X by

$$H_\infty^\varepsilon(X) = \max_{X' : X' \approx_\varepsilon X} H_\infty(X').$$

A random variable X is an (n, k) source if X is distributed over $\{0,1\}^n$ and has min-entropy at least k. When n is clear from the context we sometimes omit it and simply say that X is a k-source. Every k-source X can be expressed as a convex combination of flat distributions each with min-entropy at least k.

Definition 2.12 (average conditional min-entropy). *Let X, Y be two random variables. The* average conditional min-entropy *is defined by*

$$\widetilde{H}_\infty(X|Y) = -\log\left(\mathbb{E}_{y \sim Y}\left[2^{-H_\infty(X|Y=y)}\right]\right).$$

Lemma 2.13. *Let X, Y be two random variables such that $|\mathrm{Supp}(Y)| \leq 2^\ell$. Then, $\widetilde{H}_\infty(X|Y) \geq H_\infty(X) - \ell$.*

2.4 Condensers, Extractors and Samplers

We start by defining extractors and condensers.

Definition 2.14 (extractor). *A function*

$$\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

is a (k, ε) seeded extractor if the following holds. For every (n, k) source X, $\mathsf{Ext}(X, Y) \approx_\varepsilon U_m$, where Y is uniformly distributed over $\{0,1\}^d$ and is independent of X. We say Ext is a strong (k, ε) seeded extractor if $(\mathsf{Ext}(X, Y), Y) \approx_\varepsilon (U_m, Y)$.

Indeed, whenever a source X has sufficient min-entropy and is independent of the seed Y, we are guaranteed that a strong seeded extractor Ext gives us $(\mathsf{Ext}(X, Y), Y) \approx_\varepsilon (U_m, Y)$. The following lemma shows that $(\mathsf{Ext}(X, Y), Y)$ is nearly independent of any random variable $\mathcal{H}$, such that X, Y are independent conditioned on $\mathcal{H}$ and $X|\mathcal{H}$ has sufficient min-entropy.

Lemma 2.15 ([DORS08, CS16]). *Let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a strong (k, ε) extractor. Let X be distributed over $\{0,1\}^n$ and let $\mathcal{H}$ be some random variable such that $\widetilde{H}_\infty(X|\mathcal{H}) \geq k + \log \frac{1}{\varepsilon}$. Let Y be uniformly distributed over $\{0,1\}^d$, independent of X conditioned on $\mathcal{H}$. Then,*

$$(\mathsf{Ext}(X, Y), Y, \mathcal{H}) \approx_{2\varepsilon} (U_m, Y, \mathcal{H}).$$

Definition 2.16 (condenser). *A function*

$$\mathsf{Cond}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

is a (k, k', ε) condenser if the following holds. For every (n, k) source X, $H_\infty^\varepsilon(\mathsf{Cond}(X, Y)) \geq k'$, where Y is uniformly distributed over $\{0,1\}^d$ and is independent of X. We say Cond is a strong (k, k', ε) condenser if $(\mathsf{Cond}(X, Y), Y)$ is ε-close to some (D, Y) having min-entropy $d + k'$.

We proceed by defining *density samplers*.

Definition 2.17 (sampler)**.** *Let $\Gamma \colon [N] \times [D] \to [M]$.*

- *We say $x \in [N]$ is ε-*bad for *$B \subseteq [M]$ if*
$$\left| \Pr_{y \sim U_{[D]}} [\Gamma(x,y) \in B] - \mu(B) \right| > \varepsilon.$$
- *We say Γ is a (δ, ε) sampler if for every $B \subseteq [M]$ we have that*
$$|\{x \in [N] : x \text{ is } \varepsilon\text{-bad for } B\}| < \delta N.$$

We often reason about samplers via their bipartite graph interpretation. That is, we interpret the function $\Gamma \colon [N] \times [D] \to [M]$ as a bipartite graph (V, E) on vertex set $V = [N] \cup [M]$, where $(i,j) \in E$ if and only if there exists $k \in [D]$ such that $\Gamma(i,k) = j$. We refer to $[N]$ as the "left" vertex set and $[M]$ as the "right" vertex set. Finally, by abuse of notation, we denote $\Gamma(i)$ for $i \in [N]$ as the set of vertices in $[M]$ connected to i in the bipartite graph. Similarly, we denote $\Gamma^{-1}(j)$ for $j \in [M]$ as the set of vertices in $[N]$ connected to j. Finally, if the bipartite graph corresponding to Γ is biregular with right degree a, we use $\Gamma^{-1}(j, \ell)$ for $j \in [M]$ and $\ell \in [a]$ to denote the ℓ-th neighbor of j (under an arbitrary ordering of $[N]$).

Lemma 2.18 ([Zuc97])**.** *Let $\mathsf{Ext} \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k, ε) extractor. Then, Ext is also a $(\delta = 2^{k-n}, \varepsilon)$ sampler.*

2.4.1 The Random-Walk Sampler

When the min-entropy k we wish to support is very close to n, one can use the following simple construction based on random walks on expanders. Before giving the construction, we will need fully explicit expanders over any number of vertices. The following theorem is implied by the constructions of [GG81, RVW02, MRSV19].

Definition 2.19 (expander graph)**.** *We say that an undirected regular graph G is a λ-expander if all eigenvalues of the normalized adjacency matrix of G other than 1 are at most λ in absolute value.*

Theorem 2.20. *For every constant $0 < \lambda < 1$ there exists a constant integer $d = d(\lambda)$ such that the following holds. For every positive integer n there exists a connected d-regular undirected graph which is a λ-expander. Given a vertex $x \in [n]$ and an edge label $i \in [d]$, the i-th neighbor of x can be computed in time* $\mathrm{polylog(n)}$.

We are given m, t and $\varepsilon > 0$. Let G be a regular λ-expander over 2^m vertices for some $\lambda < \frac{1}{2}$ and constant degree $d = 2^c$. Let $n = m + c(t-1)$ and consider the function $\mathsf{Ext} \colon \{0,1\}^n \times [t] \to \{0,1\}^m$ such that for $x = (v_1, i_1, \dots, i_{t-1}) \in \{0,1\}^n$ and $y \in [t]$, $\mathsf{Ext}(x,y)$ outputs the y-th vertex in the walk $v_1 \sim_{i_1} v_2 \sim_{i_2} \sim \dots \sim_{i_{t-1}} v_t$ over G. The fact that Ext is an extractor follows from the expander Chernoff bound.

Theorem 2.21 ([Zuc07], following [Gil98, Hea08])**.** *For every m and t, let n and Ext be as above. Then, for every $\varepsilon > 0$ and $k \geq n - \frac{\varepsilon^2 t}{8}$, Ext is a (k, ε) extractor.*

Using Lemma 2.18, the above theorem implies a sampler with good parameters when we aim for relatively high δ and ε (say, both being constants).

Corollary 2.22. *There exists a constant $c \geq 1$ such that the following holds. For every positive integer M and $\varepsilon, \delta > 0$ there exists a (δ, ε) sampler $\Gamma \colon [N] \times [D] \to [M]$ where $D = \frac{8}{\varepsilon^2} \log \frac{1}{\delta}$ and $N = 2^{c(D-1)} M$. Moreover, Γ is biregular (i.e., the corresponding bipartite graph has left-degree t and right-degree $a = \frac{ND}{M}$).*

For our application, we wish to fix N and fix the right-degree of Γ. Doing so requires us to choose D such that $D2^{c(D-1)} = a$. We note that this implies so $\frac{\log a}{c} - \frac{\log \log a}{c} < D - 1 < \frac{\log a}{c}$, so we have that $M = \frac{N}{2^{c(D-1)}} = O(\frac{N \log a}{a})$. We record this result in the following corollary, together with a bound on the time it takes to compute Γ and Γ^{-1}.

Corollary 2.23. *There exists a constant $c \geq 1$ such that the following holds. For every positive integers N and $a \leq N$ there exists a (δ, ε) sampler $\Gamma \colon [N] \times [D] \to [M]$ whose corresponding bipartite graph is biregular, with $M = O(\frac{N \log a}{a})$, left-degree $D = O(\log a)$ and right-degree a, whenever $\log a \geq \frac{c}{\varepsilon^2} \log \frac{1}{\delta}$.*

Moreover, for every $x \in [N]$ and $i \in [D]$, $\Gamma(x, i)$ is computable in time $\log a \cdot \operatorname{polylog}(N)$, and for every $z \in [M]$ and $j \in [a]$, $\Gamma^{-1}(z, j)$ is computable in time $\log a \cdot \operatorname{polylog}(N)$ as well.

2.5 Pseudoentropy

Next, we discuss *computational* notions of min-entropy, or, (min-) *pseudoentropy*. We start with the standard, widely used, notion of pseudoentropy due to Håstad et al. [HILL99].

Definition 2.24 (HILL pseudoentropy)**.** *Let X be a random variable distributed over $\{0,1\}^n$, an integer s and $\varepsilon > 0$. We say that $H^{\mathrm{HILL}}_{s,\varepsilon}(X) \geq k$ if there exists a random variable $Y \sim \{0,1\}^n$ with $H_\infty(Y) \geq k$ such that for every circuit $D \colon \{0,1\}^n \to \{0,1\}$ of size s, $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

A weaker notion, given by Barak, Shaltiel and Wigderson [BSW03] allows the random variable having true min-entropy to depend on the distinguisher itself.

Definition 2.25 (metric pseudoentropy)**.** *Let X be a random variable distributed over $\{0,1\}^n$, an integer s and $\varepsilon > 0$. We say that $H^{\mathrm{metric}}_{s,\varepsilon}(X) \geq k$ if for every circuit $D \colon \{0,1\}^n \to \{0,1\}$ of size s there exists $Y \sim \{0,1\}^n$ such that $H_\infty(Y) \geq k$ and $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

We say that $H^{\mathrm{Nmetric}}_{s,\varepsilon}(X) \geq k$ if we allow D to be an SVN circuit.

We record the following standard fact in pseudorandomness.

Claim 2.26. *If $X \sim \{0,1\}^n$ is a random variable such that $H^{\varepsilon}_\infty(X) \geq k$ then for every set $D \subseteq \{0,1\}^n$ it holds that $\Pr[X \in D] \leq \frac{|D|}{2^k} + \varepsilon$.*

Barak et al. also give such a result in the computational world, when we use metric pseudoentropy.

Lemma 2.27 ([BSW03])**.** *Let X be a random variable distributed over $\{0,1\}^n$, an integer s and $\varepsilon > 0$ so that $H^{\mathrm{metric}}_{s,\varepsilon}(X) \geq k$. Then, for every circuit $D \colon \{0,1\}^n \to \{0,1\}$ of size s it holds that $\Pr[D(X) = 1] \leq \frac{|\mathrm{Supp}(D)|}{2^k} + \varepsilon$. The same statement holds when we consider SVN circuits.*

2.5.1 Yao Pseudoentropy

A different definition for computational entropy was given by Yao [Yao82] who used *compression* rather than *indistinguishability*.

Definition 2.28. *For some integers n, s and $D \subseteq \{0,1\}^n$, we say that $D \in \mathcal{C}^{\mathrm{Yao}}_{\ell,s}$ if there exist circuits $c \colon \{0,1\}^n \to \{0,1\}^\ell$ and $d \colon \{0,1\}^\ell \to \{0,1\}^n$, each of size s, such that $D = \{x : d(c(x)) = x\}$. We refer to c as the compressing circuit and to d as the decompressing one.*

Definition 2.29 (Yao pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, an integer s and $\varepsilon > 0$. We say that $H^{\mathrm{Yao}}_{s,\varepsilon}(X) \geq k$ if for every $\ell < k$ and $D \in \mathcal{C}^{\mathrm{Yao}}_{\ell,s}$, $\Pr[X \in D] \leq 2^{\ell-k} + \varepsilon$.*

We say that $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$ if we allow each D to have SVN circuits for the compression and decompression.

It is not clear whether $H^{\mathrm{Yao}}_{s,\varepsilon}(X) \geq k$ implies $H^{\mathrm{HILL}}_{s',\varepsilon'}(X) \geq k'$ for some suitable k', s', ε'.[13] However, Barak et al. [BSW03] showed that if one is considering hardness against polynomial-sized circuits with NP gates, Yao pseudoentropy does imply metric pseudoentropy. We reprove their result, and here we do it for SVN circuits.

Lemma 2.30 (following [BSW03]). *There exists a constant $0 < \gamma < 1$ such that the following holds. Let X be a random variable distributed over $\{0,1\}^n$ and $\varepsilon > 0$. There exists $s_0 = \widetilde{O}(n)$ such that for every $s \geq s_0$, $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$ implies $H^{\mathrm{Nmetric}}_{\gamma s,\varepsilon}(X) \geq \frac{k}{2}$.*

Proof: Assume towards a contradiction that $H^{\mathrm{Nmetric}}_{\gamma s,\varepsilon}(X) < \frac{k}{2}$ for some universal constant $\gamma < 1$ to be determined later. Thus, by Lemma 2.27, there exists an SVN circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size γs, with $|\mathrm{Supp}(D)| < 2^{k/2}$, for which

$$\Pr[D(X) = 1] > \frac{|\mathrm{Supp}(D)|}{2^{k/2}} + \varepsilon.$$

Denote $t = \log|\mathrm{Supp}(D)| < \frac{k}{2}$ and let $\mathcal{H} \subseteq \{0,1\}^n \to \{0,1\}^{2t}$ be a two-universal family of hash functions. We recall that a two-universal family of hash functions $\mathcal{H}$ satisfies that for every distinct $x, y \in \{0,1\}^n$ and every $\sigma_1, \sigma_2 \in \{0,1\}^{2t}$, we have

$$\Pr_{h \sim \mathcal{H}}[h(x) = \sigma_1 \wedge h(y) = \sigma_2] = \left(\frac{1}{2^{2t}}\right)^2.$$

For an efficient implementation of a small $\mathcal{H}$, we can take it to be the set of all affine functions over $\mathbb{F} = \mathrm{GF}(2^n)$, and so $h^\star(x)$ amounts to computing $ax + b$ over $\mathbb{F}$ for some fixed $a, b \in \mathbb{F}$ and truncating the last $n - 2t$ digits. Arithmetics in $\mathbb{F}$ can be done by circuits of size $\widetilde{O}(n)$.

For a random $h \sim \mathcal{H}$ and distinct $x, y \in \{0,1\}^n$, let $I_{x,y}$ be the indicator random variable which is 1 if and only if $h(x) = h(y)$. By the properties of $\mathcal{H}$,

$$\mathbb{E}\left[\sum_{\{x,y\} \subseteq D} I_{x,y}\right] \leq \binom{2^t}{2} \Pr[I_{x,y} = 1] \leq \binom{2^t}{2} 2^{-2t} < 1,$$

so there exists some $h^\star \in \mathcal{H}$ which is one to one on $\mathrm{Supp}(D)$.

Set $\ell = 2t$, and let $c\colon \{0,1\}^n \to \{0,1\}^\ell$ be the compressing circuits, which simply computes $h^\star$. For a decompressing circuit that gets $z \in \{0,1\}^\ell$ as an input, we should find the unique $x \in \mathrm{Supp}(D)$ such that $h^\star(x) = z$. Namely, consider the SVN circuit $d\colon \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^n$ that simply returns $d(z,x) = x$ and outputs $d(z,x)_{\mathsf{check}} = 1$ if and only if both $D(x) = 1$ and $h^\star(x) = z$. Evaluating $h^\star(x)$ can be done by a circuit of size $s_0 = \widetilde{O}(n)$ and the constant γ can be set so that both c and d are of size s.

Finally, note that

$$\Pr[D(X) = 1] > 2^{t - \frac{k}{2}} + \varepsilon > 2^{\ell - k} + \varepsilon,$$

which contradicts the fact that $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$. ■

[13]Hsiao, Lu and Reyzin [HLR07] studied *conditional* versions of HILL and Yao pseudoentropies and proved that under these definitions the above implication does not hold. Furthermore, they managed to extract more pseudorandom bits using Yao pseudoentropy based techniques than seems possible from HILL pseudoentropy. Wee gave an oracle under which the Yao pseudoentropy is larger than the HILL pseudoentropy by a factor of roughly 2 [Wee04].

2.6 Pseudoentropy Generators and Pseudorandom Generators

We say that a distribution $X \sim \{0,1\}^n$ *ε-fools* a circuit D with n inputs if $D(X) \approx_\varepsilon D(U_n)$. A pseudorandom generator against a class $\mathcal{C}$ is a function whose output distribution fools any function from $\mathcal{C}$.

Definition 2.31 (PRG). *Let $\mathcal{C} \subseteq \{0,1\}^n \to \{0,1\}$ be a class of functions. We say that $G\colon \{0,1\}^n \to \{0,1\}$ ε-fools $\mathcal{C}$ (or, is an ε-*PRG against *$\mathcal{C}$) if for every $C \in \mathcal{C}$,*

$$|\mathbb{E}[C(G(U_d))] - \mathbb{E}[C(U_n)]| \leq \varepsilon.$$

When $\mathcal{C}$ is the class of functions computable by circuits of size s, we say that G ε-fools circuits of size s.

In this work, we will also construct weaker variants. When the output of a generator is not pseudorandom but does have high pseudoentropy, we say that the generator is a *pseudoentropy generator*. Our pseudoentropy generators will output random variables having high *metric* pseudoentropy.

Definition 2.32 (metric PEG). *We say that $G\colon \{0,1\}^d \to \{0,1\}^n$ is a (k,s,ε) metric pseudoentropy generator (PEG) if*

$$H^{\text{metric}}_{s,\varepsilon}(G(Y)) \geq k,$$

where Y is the uniform distribution over d bits. We say that G is a strong (k,s,ε) metric pseudoentropy generator if $H^{\text{metric}}_{s,\varepsilon}(Y \circ G(Y)) \geq k$. When the output has high Nmetric pseudoentropy, we say it is an SVN pseudoentropy generator.

One can show that for $k = n$ the above definition coincides with the standard definition of PRGs [BRSW12].

3 A Locally List Recoverable Code From Local List Decoding

In this section, we construct locally recoverable codes useful for constructing a pseudoentropy generator. For this purpose, for some sufficiently small constant $\alpha > 0$ and any constant $\varepsilon < 1$, we wish to construct a code $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ that is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for at most $Q = O(n^{O(\alpha)})$ queries, list size $\ell = 2^{n^{1-O(\alpha)}}$, decoding circuit size $s_{\mathcal{C}} \leq n^{1-\alpha}$, and alphabet size $|\Sigma| = 2^{n^{1-O(\alpha)}}$ while maintaining a relatively high rate (specifically, $m = n^{O(\alpha)}$).

To this end, we utilize our nonuniform definition of list recovery and in fact construct a circuit that takes as advice a pointer for each list $S_1, \ldots, S_m$, as these are independent of the specific coordinate we wish to decode. The pointers then induce a string in Σ^m to which the circuit has oracle access. Thus, the circuit we construct only needs to efficiently list decode the induced string. Following [STV01], we apply the list decoding properties of the Reed-Muller code. More specifically, we use the following theorem.

Theorem 3.1 ([STV01]). *Let $\mathcal{C}_{\mathsf{RM}}$ denote the $\left[q^t, \binom{t+d}{d}, 1 - \frac{d}{q}\right]_q$ Reed-Muller code. That is, the code consisting of all t-variate polynomials of total degree d over $\mathbb{F}_q$. Then for any constant $1 > \zeta > 0$, $\mathcal{C}_{\mathsf{RM}}$ is locally list decodable using $Q = O_\zeta(q^2)$ queries, handling $2\sqrt{2}\sqrt{\frac{d}{q}}$ fraction of agreement, with failure probability ζ, circuit size $s = \mathrm{poly}(t,q)$, and output list size $L = O(q^t)$. For a constant t, one can achieve $s = \widetilde{O}_t(q^4)$.*

To see why one can achieve $s = \widetilde{O}_t(q^4)$, we briefly touch upon the details in [STV01]. The decoding circuit for the Reed-Muller code works by performing list decoding for *univariate* polynomials along q lines in $\mathbb{F}_q^t$. The latter can be done by first solving a system of at most $O(q)$ linear equations in $O(q)$ unknowns over $\mathbb{F}_q$ (which takes time $O(q^3)$) to find a certain bivariate polynomial over $\mathbb{F}_q$. Next, the algorithm finds roots of the bivariate polynomial [Sud97]. Finding such roots requires at most $O(q)$ applications of factoring univariate polynomials, which takes time at most $O(q^2)$ by fastest known results [KU11, ALRS98]. Thus overall, for constant t, the size of the decoding circuit s is at most $\widetilde{O}(q(q^3 + q^3)) = \widetilde{O}(q^4)$.

3.1 The Construction

Our construction is a variant of an expander-based code by Alon et al. [ABN+92], except we use samplers instead of expanders. Given a positive integer n and constants $0 < \varepsilon < 1$ and $0 < \alpha < \frac{1}{6}$, we construct $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ that is locally list recoverable by a small circuit using $\widetilde{O}(n^{O(\alpha)})$ queries, with input list size $\ell = 2^{n^{1-O(\alpha)}}$. The construction is as follows.

- In Theorem 3.1, set $\zeta = \frac{1}{6}$, $t = \frac{1}{\alpha}$ and $q = \frac{32}{\varepsilon^2} \cdot d$ with d chosen so that $n = \binom{t+d}{d} \log q$. Working out the parameters, one can see that $d = \widetilde{O}_{\varepsilon,\alpha}(n^\alpha)$ and $q = \widetilde{O}_{\varepsilon,\alpha}(n^\alpha)$ (here, the ε, α subscripts hide a $\frac{1}{\alpha}$ and $\frac{1}{\varepsilon^2}$ factor respectively). The resulting code

  $$\mathcal{C}_{\mathsf{RM}}\colon \{0,1\}^n \to \mathbb{F}_q^{\bar{n}=cn}$$

  is a

  $$\left(Q = \widetilde{O}_{\varepsilon,\alpha}(n^{2\alpha}), \frac{\varepsilon}{2}, \zeta = \frac{1}{6}, s_{\mathsf{RM}} = \widetilde{O}(n^{4\alpha}), L = O(n)\right)$$

  locally list decodable code for $c = \left(\frac{32}{\varepsilon^2 \alpha}\right)^{\frac{1}{\alpha}} \log n$.

- Set $a = n^{1-5\alpha}$. For $m = O\left(\frac{\bar{n}\log a}{a}\right) = \widetilde{O}_{\varepsilon,\alpha}(n^{5\alpha})$, let

  $$\Gamma\colon [\bar{n}] \times [D] \to [m]$$

  be a biregular $(\frac{1}{3}, \frac{\varepsilon}{8})$ sampler guaranteed to us by Corollary 2.23, with right-degree a and left-degree $D = \frac{ma}{\bar{n}} = O(\log n)$.

We are now prepared to construct our desired list recoverable code. Define

$$\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$$

where $\Sigma = \mathbb{F}_q^a$, so that for every $f \in \{0,1\}^n$ and $z \in [m]$,

$$\mathcal{C}(f)_z = \mathcal{C}_{\mathsf{RM}}(f)_{\Gamma^{-1}(z,1)} \circ \ldots \circ \mathcal{C}_{\mathsf{RM}}(f)_{\Gamma^{-1}(z,a)}.$$

See also Figure 1. Note that the rate of $\mathcal{C}$ is given by $\frac{n}{ma\log q} = \Omega_\alpha\left(\frac{1}{\log^2 n}\right)$.

We note that we do not make use of any special property of the Reed-Muller code, and in fact any efficient locally list decodable code that has relatively high rate and handles arbitrarily small fraction of agreement will work just as well. In particular, we made no further attempt to reduce the alphabet size q or the decoding runtime, which dictates how small we must take α to be. Moreover, improving the dependence of Q on ε may lead to better results for non-constant ε.

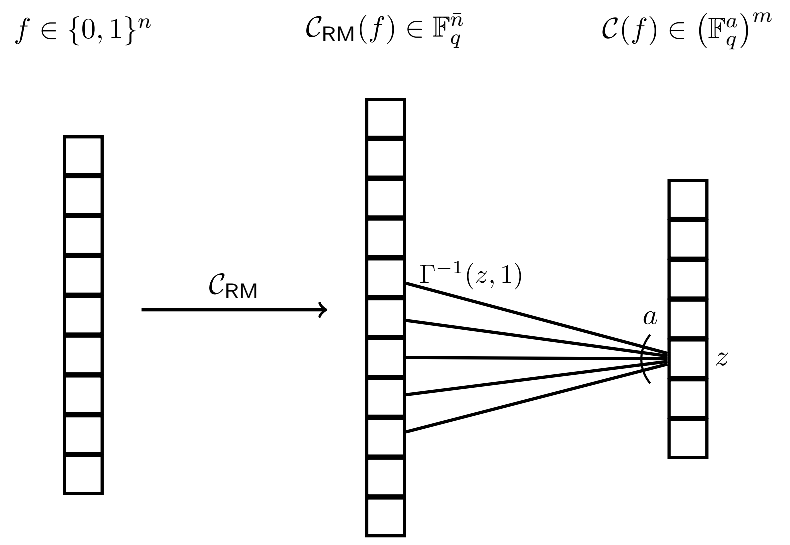
$f \in \{0,1\}^n$
$\mathcal{C}_{\mathsf{RM}}(f) \in \mathbb{F}_q^{\bar{n}}$
$\mathcal{C}(f) \in \left(\mathbb{F}_q^a\right)^m$
$\mathcal{C}_{\mathsf{RM}}$
$\Gamma^{-1}(z,1)$
a
z

Figure 1: The construction of our locally recoverable code. The middle layer is the Reed-Muller encoding of f with symbols in $\mathbb{F}_q$. A symbol at coordinate z in our final code is an element of $\mathbb{F}_q^a$, where the a elements of $\mathbb{F}_q$ are determined by $\Gamma^{-1}(z)$.

3.2 Analysis

Theorem 3.2. *For any positive integer n, any constants $0 < \varepsilon < 1$ and $0 < \alpha \le \frac{1}{6}$, the code $\mathcal{C}$ constructed above is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\ell = 2^{n^{1-7\alpha}}$ and $s_{\mathcal{C}} = n^{1-\alpha}$.*

Proof: To describe our local list recovery algorithm, we give a family of circuits, indexed by advice, one of which successfully decodes. We describe our decoding algorithm when given the correct advice, which shows that one circuit successfully decodes.

Towards this end, fix any codeword $\mathcal{C}(f)$ for $f \in \{0,1\}^n$ for which $\mathcal{C}(f)_z \in S_z$ for at least ε-fraction of $z \in [m]$. We begin by describing a *randomized* circuit for computing $\mathcal{C}(f)$. Consider

$$A_{adv}^{\bar{S}}(x, r)$$

which takes as input $x \in [n]$, uses randomness string r, has oracle access to lists $\bar{S} = S_1, \ldots, S_m$, and aims to compute f_x. The decoder circuit $A_{adv}^{\bar{S}}$ implements the following procedure.

1. We first set up some preliminaries.
   - $A_{adv}^{\bar{S}}$ is hardwired with advice $adv = (y_1, \ldots, y_m, \mathsf{RM})$, where each $y_z \in \{0,1\}^{\log \ell}$ points to some list entry in the z-th list. We assume the correct advice is given. Specifically, for each z such that $\mathcal{C}(f)_z \in S_z$, let the correct y_z point to $\mathcal{C}(f)_z$, otherwise let y_z be arbitrary. Note that the size of the advice is $m \log \ell = \widetilde{O}(n^{1-2\alpha})$, and only depends on f and $\bar{S}$ and not on the input x.
   - The rest of the advice RM points to a decoding circuit for the Reed-Muller code, and we assume the correct circuit is given. That is, let A_{RM} be the randomized circuit that computes f_x for every x, with probability at least $\frac{5}{6}$ when given oracle access to a word in $\mathbb{F}_q^{\bar{n}}$ with at most $1 - \frac{\varepsilon}{2}$ fraction of errors from $\mathcal{C}_{\mathsf{RM}}(f)$.
   - Write the randomness string r as $r = (r_{\mathsf{RM}}, r_\Gamma)$ for r_{RM} and r_Γ independent from each other. r_{RM} denotes the randomness required by A_{RM}. We let $r_\Gamma = (r_{\Gamma,1}, \ldots, r_{\Gamma,Q})$, where

$r_{\Gamma,j} \in [D]$ for each $j \in [Q]$. This string denotes the additional randomness of our decoding circuit.

2. Run the decoding circuit $A_{\mathsf{RM}}(x, r_{\mathsf{RM}})$. For every $j \in [Q]$, supply the answer to the j-th oracle query that A_{RM} makes to some coordinate $i \in [\bar{n}]$ as follows:

    (a) Let $z = \Gamma(i, r_{\Gamma,j})$.

    (b) Use the advice string y_z and oracle access to $\bar{S}$ to query the element $\widetilde{v} = S_z(y_z) \in \mathbb{F}_q^a$.

    (c) Return the symbol $\widetilde{v}_h \in \mathbb{F}_q$ for $h \in [a]$ such that $\Gamma^{-1}(z, h) = i$ as the guess for the i-th coordinate for A_{RM}.

3. Return the output of $A_{\mathsf{RM}}(x, r)$.

First, we claim:

Claim 3.3. *$A_{adv}^{\bar{S}}$ can be computed by a circuit of size*

$$\widetilde{O}(n^{1-2\alpha})$$

which makes at most $Q = \widetilde{O}(n^{2\alpha})$ oracle queries to $\bar{S}$.

Proof: The inputs to the circuit are $x \in \{0,1\}^{\log n}$ and the randomness r. The number of queries equals the number of queries of the local decoding algorithm A_{RM}. Since A_{RM} makes at most $Q = \widetilde{O}(n^{2\alpha})$ queries to $[\bar{n}]$, and for each such query we pick a random neighbor in Γ among $D = O(\log n)$ neighbors, the total amount of randomness required for r_Γ is $\widetilde{O}(n^{2\alpha})$. The advice that we hardwire to the circuit has length $\widetilde{O}(n^{1-2\alpha})$.

Next, the size of A_{RM} itself is $\widetilde{O}(n^{4\alpha})$, and for each of the $Q = \widetilde{O}(n^{2\alpha})$ queries, we compute a neighbor in Γ which by Corollary 2.23 takes time $\log \bar{n} \cdot \mathrm{polylog}(n^{1-3\alpha})$ (and hence requires a circuit of size at most $\mathrm{polylog}\, n$). Overall, the total size of the circuit is

$$\log n + \widetilde{O}(n^{2\alpha}) + \widetilde{O}(n^{1-2\alpha}) + \widetilde{O}(n^{4\alpha}) + \widetilde{O}(n^{2\alpha})\,\mathrm{polylog}\, n = \widetilde{O}(n^{1-2\alpha}),$$

where we used the fact that $\alpha \le \frac{1}{6}$. ■

We shall now prove the correctness of $A_{adv}^{\bar{S}}$.

Lemma 3.4. *Fix f and $\bar{S} = S_1, \ldots, S_m$ as above. Then for every $x \in [n]$:*

$$\Pr_{r \sim R}\left[A_{adv}^{\bar{S}}(x, r) = f(x)\right] \ge \frac{2}{3}.$$

Proof: The idea is to show that the queries A_{RM} are essentially queries to a word with agreement at least $\frac{\varepsilon}{2}$ with $\mathcal{C}_{\mathsf{RM}}(f)$. To this end, first consider an alternate version of $A_{adv}^{\bar{S}}$, denoted $A_{\mathsf{alt},adv}^{\bar{S}}$, which instead uses a randomness string $r = (r_{\mathsf{RM}}, r_\Gamma)$ where now $r_\Gamma = (r_{\Gamma,1}, \ldots, r_{\Gamma,\bar{n}})$ for $r_j \in [D]$. Instead of picking a random neighbor in the sampler for each query made by A_{RM}, the alternate version first picks a random neighbor $\Gamma(i)$ for every $i \in [\bar{n}]$. Then, for any queries on coordinate i, the answer to the query is supplied by using the neighbor $r_{\Gamma,i}$. Notice that a fixed choice of r_Γ induces a message $w \in \mathbb{F}_q^{\bar{n}}$ in the natural way, where w_i is the appropriate symbol of $S_{\Gamma(i,r_{\Gamma,i})}(y_{\Gamma(i,r_{\Gamma,i})})$.

We first prove that with probability at most $\frac{1}{6}$ over $r_\Gamma = (r_1, \ldots, r_{\bar{n}})$ the induced codeword $w(r_\Gamma)$ has less than $\frac{\varepsilon}{2}$ agreement with $\mathcal{C}_{\mathsf{RM}}(f)$. Let $\mathbf{G} \subseteq [m]$ be the set of good lists, of density at least ε, for which $\mathcal{C}(f)_z$ is contained in S_z. That is:

$$\mathbf{G} = \{z \in [m] : \mathcal{C}(f)_z \in S_z\}$$

Observe that for any $i \in [\bar{n}]$ for which $\Gamma(i, r_{\Gamma,i}) \in \mathbf{G}$, the coordinate of the induced codeword, w_i, agrees with $\mathcal{C}_{\mathsf{RM}}(f)_i$ (in other words A_{RM}'s query on i will be correct). Now let $\mathbf{B} \subset [\bar{n}]$ be the set of bad coordinates of $\mathcal{C}_{\mathsf{RM}}$, for which less than $\frac{7\varepsilon}{8}$ of its neighbors are in $\mathbf{G}$. Namely,

$$\mathbf{B} = \left\{ i \in [\bar{n}] : \mu_{\Gamma(i)}(\Gamma(i) \cap \mathbf{G}) < \frac{7\varepsilon}{8} \right\}.$$

By the sampler property, we know that $\mu(\mathbf{B}) \leq \frac{1}{3}$. Now note that for any $i \notin \mathbf{B}$, the probability that $\Gamma(i, R_{\Gamma,i}) \in \mathbf{G}$ is at least $\frac{7\varepsilon}{8}$. Since each $R_{\Gamma,i}$ is independent, by a Chernoff bound (over a sample size at least $\frac{2}{3}\bar{n}$), the probability that less than $\frac{3\varepsilon}{4}$ fraction of i-s outside of $\mathbf{B}$ have $w_i = \mathcal{C}_{\mathsf{RM}}(f)_i$ is at most $\frac{1}{6}$. Thus with probability at least $\frac{5}{6}$ over the randomness r_Γ, we have that w agrees with $\mathcal{C}_{\mathsf{RM}}(f)$ in at least

$$(1 - \mu(\mathbf{B}))\,\frac{3\varepsilon}{4} \geq \left(1 - \frac{1}{3}\right)\frac{3\varepsilon}{4} = \frac{\varepsilon}{2}$$

fraction of coordinates.

This means that with probability at least $\frac{5}{6}$, A_{RM} has oracle access to a word w with at least $\frac{\varepsilon}{2}$ agreement with $\mathcal{C}_{\mathsf{RM}}(f)$. In this case, by definition, the probability that A_{RM} outputs an incorrect symbol is at most $\frac{1}{6}$. and thus

$$\Pr_{r \sim R}\left[A^{\bar{S}}_{\mathsf{alt},adv}(x, r) = f(x)\right] \geq \frac{2}{3}.$$

As a final observation, we note that the behavior of $A^{\bar{S}}_{\mathsf{alt},adv}$ and the original $A^{\bar{S}}_{adv}$ must be identical. This is because the randomness $r_{\Gamma,i}$ for any query on coordinate $i \in [\bar{n}]$ can be supplied when required, and that any $r_{\Gamma,i}$ for an index i that is not queried by the algorithm has no effect on its outcome. ■

So far we constructed a randomized circuit that computes $\mathcal{C}(f)$ with high probability, so what remains is constructing a deterministic counterpart. This follows from a standard amplification argument. Lemma 3.4 tells us that the randomized circuit $A^{\bar{S}}_{adv}$ is incorrect with probability at most $\frac{1}{3}$. We can thus construct a new circuit that makes $M = O(\log n)$ repeated runs of $A^{\bar{S}}_{adv}$ with independent randomness and takes the majority vote. By Chernoff, the error probability of the new amplified circuit is at most $\frac{1}{2n}$ and so there is a fixing of the randomness string so that the amplified circuit computes f on all indices. Since the size of the unamplified circuit was $\widetilde{O}(n^{1-2\alpha})$, the final circuit is of size $\widetilde{O}(n^{1-2\alpha} \cdot M) \leq n^{1-\alpha}$. Moreover, the number of queries is $\widetilde{O}(n^{2\alpha} \cdot M) = \widetilde{O}(n^{2\alpha})$. ■

We finish this section with a calculation of the time required to compute a given coordinate of our code $\mathcal{C}$, which relates directly to the runtime of our final PRGs.

Claim 3.5. *Given $z \in [m]$, and $f \in \{0,1\}^n$, computing $\mathcal{C}(f)_z$ can be done in time $\widetilde{O}(n^2)$. Moreover, given $f \in \{0,1\}^n$, computing the entire codeword $\mathcal{C}(f)$ can be done in time $\widetilde{O}(n^2)$ as well.*

Proof: In order to compute $\mathcal{C}(f)_z$, we compute the Reed-Muller code at $n^{1-5\varepsilon}$ points determined by the sampler Γ. To do so, we first recover the coefficients of the Reed-Muller polynomial. Recall that in the standard systematic Reed-Muller encoding, we interpret the message $f \in \{0,1\}^n$ as values of a polynomial on $H^t \subseteq \mathbb{F}_q^t$ for some fixed $H \subseteq \mathbb{F}_q$ of size roughly $|H| = \frac{d}{t}$. We then write the polynomial of degree at most d that agrees with the message f as

$$p(x_1, \ldots, x_t) = \sum_{(a_1,\ldots,a_t) \in H^t} \prod_{i \in [t]} \delta_{a_i}(x_i)$$

for univariate polynomials $\delta_a \colon \mathbb{F}_q \to \mathbb{F}_q$, where for each $a \in H$,

$$\delta_a(x) = \prod_{b \in H \setminus \{a\}} \frac{x-b}{a-b}.$$

We can write down the coefficients of each δ_a by first writing down δ_a in the form above and expanding the polynomial. For any fixed a, we can do so in time $\widetilde{O}(|H|)^2) = \widetilde{O}(n^{2\alpha})$ using a $\widetilde{O}(d)$ time algorithm for multiplying two univariate polynomials of degree d. We can then can expand and simplify the formula for p in $O((H^t)^2) = \widetilde{O}(n^2)$ time.

Once we have the coefficients of the Reed-Muller code, we can then compute $\Gamma^{-1}(z,i) \in \mathbb{F}_q^t$ for every $i \in [a]$ for the right-degree of the sampler $a = n^{1-5\alpha}$. Each computation of Γ^{-1} requires $O(\text{polylog}\, n)$ time. Given the coefficients of the Reed-Muller polynomial, and $N = n^{1-5\alpha}$ points in $\mathbb{F}_q^t$, we invoke the Kedlaya-Umans algorithm for fast multivariate multipoint evaluation [KU11] running in time

$$O\left(t\left(\frac{d^t}{t} + q^t + N\right)\text{poly}(\log q)\right) = \widetilde{O}(n).$$

In total, the runtime is dominated by the $\widetilde{O}(n^2)$ time required to find the coefficients of the Reed-Muller polynomial.[14]

In order to compute the entire code $\mathcal{C}(f)$, we can instead invoke the fast multivariate multipoint evaluation on all $\bar{n} = \widetilde{O}(n)$ points in $\mathbb{F}_q^t$ given the coefficients of the polynomial. We can then iterate through every $z \in [m]$ with $m = \widetilde{O}(n^{5\alpha})$ and use the memoized values of the polynomial at every point to compute the appropriate symbol $\mathcal{C}(f)_z$. ■

4 A Metric Pseudoentropy Generator From Worst-Case Hardness

In this section we show how to construct a metric pseudoentropy generator from a function f that is hard for SVN circuits. The idea is to show that if a code $\mathcal{C}$ is locally list recoverable, then the symbol at a random coordinate of $\mathcal{C}(f)$ has high Yao pseudoentropy. This then implies that it must also have high metric pseudoentropy by Lemma 2.30.

Let $f \colon \{0,1\}^{\log n} \to \{0,1\}$ be such that every SVN circuit computing f has size at least $n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. That is, $\text{size}_{SVN}(f) \geq n^{1-\alpha_0}$. Let

$$\mathcal{C} \colon \{0,1\}^n \to \Sigma^m$$

be some error correcting code. Define $G^f \colon [m] \to \Sigma \equiv \{0,1\}^{\log|\Sigma|}$ so that

$$G^f(z) = \mathcal{C}(f)_z,$$

where we identify f by its truth-table in $\{0,1\}^n$.

Theorem 4.1. *Keeping the above notation, let ε, α be constants such that $\varepsilon > 0$ and $\alpha_0 < \alpha \leq \frac{1}{6}$. Assume $\mathcal{C}$ is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable so that $s_{\mathcal{C}} = O(n^{1-\alpha})$, and $\frac{n^{1-\alpha}}{Q} \geq s_0$ for some $s_0 = \widetilde{O}(\log|\Sigma|)$. Then, G^f is a strong (k, s, ε) Nmetric PEG for $k = \frac{\log \ell}{2}$ and $s = O\left(\frac{n^{1-\alpha}}{Q}\right)$.*

[14]We note that a more efficient way to prepare a representation of the Reed-Muller polynomial for fast multipoint evaluation would drastically improve the runtime of our encoding procedure.

Proof: Let Z be the uniform distribution over $[m]$. We first show that

$$H^{\text{NYao}}_{s',\varepsilon}\left(Z \circ G^f(Z)\right) \geq k'$$

for $s' = \frac{n^{1-\alpha}}{Q}$ and $k' = \log \ell$.

Assume towards a contradiction that $H^{\text{NYao}}_{s',\varepsilon}(Z \circ G^f(Z)) < k'$. Then there exists a $D \subseteq [m] \times \Sigma$, with $D \in \mathcal{C}^{\text{NYao}}_{k',s'}$ such that

$$\Pr[Z \circ G^f(Z) \in D] > \varepsilon.$$

As $D \in \mathcal{C}^{\text{NYao}}_{k',s'}$, there exist compressing and decompressing SVN circuits $c\colon [m] \times \Sigma \times \{0,1\}^w \to \{0,1\}^{k'}$ and $d\colon \{0,1\}^{k'} \times \{0,1\}^w \to [m] \times \Sigma$ of size s' such that for every $(y, \sigma) \in D$ we have $d(c(z,\sigma)) = (z, \sigma)$.

Consider the sets $S_1, \dots, S_m \subseteq \Sigma$ defined as follows:

$$S_z = \{\sigma \in \Sigma : (z, \sigma) \in D\}.$$

Notice that $\sum_{z\in[m]} |S_z| = 2^{k'} = \ell$. Moreover, for at least ε fraction of z's we know that $C(f)_z \in S_z$. Thus by definition of our list recoverable code, there exists a circuit A computing f that is of size at most $s_{\mathcal{C}}$, and uses at most Q oracle queries to the lists S_z.

We can replace each oracle query in A with a circuit that computes the SVN decompressor circuit on the appropriate string in $\{0,1\}^{k'}$ and appropriate witness string in $\{0,1\}^w$, and outputs the resulting symbol. In all, the final SVN circuit will use $\{0,1\}^{wQ}$ bits as a witness string, where each block of w bits serves as the witness to d for one of the Q oracle queries. The resulting circuit will output 1 as the check bit if and only if the check bit of all Q copies of the decompressing circuit d is 1. Since the size of the decompressing circuit is at most s', we have a final circuit of size

$$O(s_{\mathcal{C}} + Q \cdot s') = O(n^{1-\alpha})$$

that computes f, which is a contradiction of the hardness of f.

Knowing that $H^{\text{NYao}}_{s,\varepsilon}\left(Z \circ G^f(Z)\right) \geq k'$, and $s' = \widetilde{\Omega}(\log|\Sigma|)$, applying Lemma 2.30 proves the theorem. ■

Combining Theorem 3.2 with the above result gives the following.

Theorem 4.2. *For a constant $\varepsilon > 0$ and every positive integer n the following holds. Assume*

$$f\colon \{0,1\}^{\log n} \to \{0,1\}$$

is such that $\mathsf{size}_{SVN}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{6}$. Then, there exists a function

$$G^f\colon \{0,1\}^d \to \{0,1\}^m$$

that is a (k, s, ε) SVN metric PEG for $k = \frac{n^{1-7\alpha}}{2}$, $s \geq n^{1-4\alpha}$, $d = 5\alpha \log n + O(\log\log n)$ and $m = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n^2)$ to compute. Moreover, if $f \in \mathbf{DTIME}(n^{c_f})$ for some $c_f \geq 1$, the support of G^f can be computed in time $\widetilde{O}(n^{c_f+1})$.

Proof: Let $\mathcal{C}\colon \{0,1\}^n \to \Sigma^{m'}$ with $\Sigma = \mathbb{F}_q^{n^{1-5\alpha}}$, $q = \widetilde{O}(n^\alpha)$ and $m' = \widetilde{O}(n^{5\alpha})$, be the $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\ell = 2^{n^{1-7\alpha}}$ and $s_{\mathcal{C}} = O(n^{1-\alpha})$ that is guaranteed to us by Theorem 3.2. We note that $\frac{n^{1-\alpha}}{Q} = \Omega(n^{1-4\alpha}) \geq \widetilde{O}(\log|\Sigma|) = \widetilde{O}(n^{1-5\alpha})$. Thus, by applying Theorem 4.1, we get that

$$G^f : \{0,1\}^d \to \{0,1\}^m$$

is a (k, s, ε) metric PEG with $d = \log m' = 5\alpha \log n + O(\log\log n)$ and $m = \log|\Sigma| = \widetilde{O}(n^{1-5\alpha})$ for $k = \Omega(n^{1-7\alpha})$ and $s = \frac{n^{1-\alpha}}{Q} \geq n^{1-4\alpha}$.

Finally, to compute the support of the PEG, we can first compute f at every point in time $O(n^{c_f+1})$. We then essentially have oracle access to f, and so by Claim 3.5 we can compute the support in time $\widetilde{O}(n^2)$. Thus, overall the time to compute the support is $\widetilde{O}(n^{c_f+1})$. ■

Note that an SVN metric PEG is also a metric PEG with the same parameters.

5 Derandomizing Algorithms That Err Rarely

In Theorem 4.2 we proved that G^f is a metric PEG. This is already useful by itself, e.g., since it allows us to derandomize algorithms that err rarely.

Claim 5.1. *Let $C\colon \{0,1\}^n \to \{0,1\}$ be a circuit of size s for which there exists $b \in \{0,1\}$ such that C evaluates to b on all but at most $B = B(n)$ of its possible inputs. Let $X \sim \{0,1\}^n$ be such that $H^{\text{metric}}_{s,\varepsilon=1/8}(X) \geq k$ for $k \geq \log B + 3$. Then,*

$$\Pr[C(X) = 1 - b] > \frac{1}{2}.$$

Proof: By Lemma 2.27,

$$\Pr[C(X) = b] \leq \frac{B}{2^k} + \frac{1}{8} \leq \frac{1}{4},$$

so the claim follows immediately. ■

Quite surprisingly, since our PEG has a very short seed, we are able to derandomize with almost no slowdown.

Lemma 5.2. *There exists a constant $c \geq 1$ such that the following holds. For positive integers n and $t \geq n$,*[15] *let $L \subseteq \{0,1\}^n$ and let $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time t for which there exists $B = B(t)$ such that for every $x \in \{0,1\}^n$,*

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] \leq \frac{B}{2^t}.$$

Let $G\colon \{0,1\}^d \to \{0,1\}^t$ be a $(k, s, \varepsilon = 1/8)$ metric PEG for $s = ct\log t$ and $k \geq \log B + 3$.

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time $2^d t + t_P$, where t_P is for computing $\operatorname{Supp}(G)$ and can be precomputed for all algorithms with running time t.

Proof: Fix some $x \in \{0,1\}^n$ and let $C_x\colon \{0,1\}^t \to \{0,1\}$ be the circuit that computes $A(x,\cdot)$, of size s. By our assumption on A, C_x outputs $1 - L(x)$ on at most B of its inputs. By Claim 5.1, we know that

$$\Pr[C_x(G(U)) = L(x)] > \frac{1}{2}.$$

[15]From here onwards, we assume $t \geq n$ only for simplicity. When $t < n$, the circuit that computes $A(x,\cdot)$ is of size $O(n\log n)$ rather than $O(t\log t)$ and s is chosen accordingly.

Hence, the standard way of constructing A_{D} would be to first compute the set

$$I = \left\{ G(z) : z \in \{0,1\}^d \right\},$$

which is independent of C so can be though of as a preprocessing step for all circuits of a certain size, and then run $A(x, y)$ for every $y \in I$. A_{D} would then return the majority vote. ■

Combining Lemma 5.2 and Theorem 4.2, we get the following corollary.

Corollary 5.3. *There exist constants $\widetilde{c} \geq 1$ such that the following holds for all positive integers n and $t \geq n$. Assume that for every positive integer m there exists a function $f\colon \{0,1\}^m \to \{0,1\}$ computable in time $\mathbf{DTIME}(2^{c_f m})$ for some $c_f \geq 1$, for which $\mathsf{size}_{SVN}(f) > 2^{(1-\alpha_0)m}$ for some constant $\alpha_0 < \frac{1}{\widetilde{c}}$. Fix any α such that $\alpha_0 < \alpha \leq \frac{1}{\widetilde{c}}$.*

Let $L \subseteq \{0,1\}^n$ and let $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time t such that for every $x \in \{0,1\}^n$,

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] \leq 2^{-t+t^{1-\widetilde{c}\alpha}}.$$

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time $t^{1+\widetilde{c}\alpha} + t_P$, where the $t_P = t^{1+c_f+O(\alpha)}$ term corresponds to a step that can be precomputed for all algorithms with running time t.

That is, the derandomization slowdown of every randomized algorithm running in time t which errs with probability at most $2^{-t+t^{1-O(\alpha)}}$, under our complexity-theoretic assumptions, is at most $t^{O(\alpha)}$.

Proof: Set $m = \log(t^{1+c\alpha})$ for $c \geq 1$ soon to be determined, $M = 2^m$, and let $f\colon \{0,1\}^{\log t^{1+c\alpha}} \to \{0,1\}$ be the guaranteed hard function. Let

$$G^f\colon \{0,1\}^d \to \{0,1\}^t$$

be the $(k, s, \varepsilon = 1/8)$ metric PEG given in Theorem 4.2 with $d \leq 6\alpha \log n$. By Theorem 4.2, we know the output of the PEG is of length $\widetilde{O}(M^{1-5\alpha})$, so we set c such that $\widetilde{O}(M^{1-5\alpha}) = t$ which gives $5 < \frac{5}{1-5\alpha} < c < \frac{6}{1-6\alpha}$, where we assume $\alpha < \frac{1}{6}$. Set $\widetilde{c} = 8c$ and set $B = 2^{t^{1-\widetilde{c}\alpha}}$. Note that

$$k = \frac{M^{1-7\alpha}}{2} = \frac{1}{2}t^{7c\alpha^2} \cdot t^{1-(c+7)\alpha} + 3 \geq \log B + 3.$$

Also,

$$s = M^{1-4\alpha} \geq t^{1+(c-4)\alpha - 4c\alpha^2} \geq t^{1+\alpha-4c\alpha^2} = \omega(t \log t),$$

since $4c\alpha^2 = \frac{\widetilde{c}\alpha^2}{2} \leq \frac{\alpha}{2}$. Finally, observe that $d \leq \widetilde{c}\alpha \log n$ and so the Corollary follows from Lemma 5.2. ■

6 Pseudoentropy Fooling Real-Valued Distinguishers

Towards derandomizing general algorithms with only an almost-linear slowdown, we need to extend our notion of pseudoentropy and handle distinguishers which output a value in $[0,1]$ rather than a Boolean value.

The motivation for doing so is that in order to get a *pseudorandom generator* (rather than a pseudoentropy generator), we want to apply a seeded extractor on top of our pseudoentropy generator. However, we only have a random variable with high *metric* pseudoentropy, which does not seem sufficient for general extraction. More specifically, Barak et al. showed how to extract from the stronger notion of HILL pseudoentropy (see Definition 2.24), or from Yao pseudoentropy, however only in the case where the extractor is a *reconstructive* one (see [BSW03] and references therein). Although our pseudoentropy generator from Section 4 does output a random variable having high Yao pseudoentropy, it seems difficult to devise a reconstructive extractor having nearly-optimal seed length.

Barak et al. [BSW03] show how one can get high HILL pseudoentropy from a random variable having high metric pseudoentropy, but their argument has two disadvantages. First, it loses quite a bit in parameters, which we cannot afford here. Second, it implicitly assumes metric pseudoentropy for *real-valued distinguishers*. However, unlike HILL pseudoentropy, different notions of metric pseudoentropy (randomized/deterministic distinguishers, $\{0,1\}/[0,1]$-valued distinguishers) do not seem to be equivalent.[16]

We would like to extract pseudorandomness without significant loss in circuit size, and we would like to do it by applying a (non-reconstructive) seeded extractor. Towards that goal, we:

1. Revisit, in this section, the notion of Yao and metric pseudoentropy, this time with respect to real-valued distinguishers. We will prove that here too, Yao pseudoentropy implies metric pseudoentropy, and that one can apply a seeded extractor on a random variable having high metric pseudoentropy without losing too much in parameters. The caveat is that we would have to assume stronger complexity-theoretic assumptions.

2. In Section 7, prove that our PEG can in fact output pseudoentropy fooling real-valued distinguishers.

3. In Section 8, construct an explicit extractor having nearly optimal seed length (for a constant error) and computable by circuits of nearly linear size.

4. Finally, in Section 9, combine all the above components to achieve an efficient derandomization and prove Theorem 9.4.

6.1 Metric* Pseudoentropy

We begin with the generalized definition of metric pseudoentropy.

Definition 6.1 (metric* pseudoentropy, [DP08, FR12])**.** *Let X be a random variable distributed over $\{0,1\}^n$, a positive integer s and $\varepsilon > 0$. We say that $H^{\text{metric}^\star}_{s,\varepsilon}(X) \geq k$ if for every circuit $D\colon \{0,1\}^n \to [0,1]$ of size s, there exists $Y \sim \{0,1\}^n$ such that $H_\infty(Y) \geq k$ and $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

We say that $H^{\mathbf{P}\text{metric}^\star}_{s,\varepsilon}(X) \geq k$ if we allow D to be a $\mathbf{FP}$-circuit, and $H^{\mathbf{N}\text{metric}^\star}_{s,\varepsilon}(X) \geq k$ if we allow D to be an $\mathbf{FNP}$-circuit.

Similar to Definition 2.32, we define pseudoentropy generators that output random variables having high metric* or its variants.

Definition 6.2 (metric* PEG)**.** *We say that $G\colon \{0,1\}^d \to \{0,1\}^n$ is a (k,s,ε) metric* pseudoentropy generator (PEG) if*

$$H^{\text{metric}^\star}_{s,\varepsilon}(G(Y)) \geq k,$$

[16] Later works address this issue explicitly and distinguishes between each notion of metric pseudoentropy (see, e.g., [CKLR11, FR12, FOR15, SGP15]).

*where Y is the uniform distribution over d bits. We say that G is a strong (k, s, ε) metric* pseudoentropy generator if $H^{\mathrm{metric}^\star}_{s,\varepsilon}(Y \circ G(Y)) \geq k$. When the output has high **P**metric* pseudoentropy or **N**metric* pseudoentropy, we say it is a **P**metric* or **N**metric* pseudoentropy generator, respectively.*

We will need an alternative characterization of metric pseudoentropy due to Skorski [Sko15].

Theorem 6.3 ([Sko15])**.** *For all positive integers s, n and every k and $\varepsilon > 0$ the following holds. Let X be a random variable distributed over $\{0,1\}^n$. Then, $H^{\mathrm{metric}^\star}_{s,\varepsilon}(X) \geq k$ if and only every circuit $D\colon \{0,1\}^n \to [0,1]$ of size s satisfies*

$$\mathbb{E}[D(X)] \leq \mathbb{E}[D(Y^\star)] + \varepsilon,$$

where $Y^\star$ is uniform over 2^k values x corresponding the largest values of $D(x)$. (Break ties arbitrarily.)

We now prove an analogue of Lemma 2.27, stating that the weight of a high metric pseudoentropy random variable inside a set computable by a small circuit cannot be too large, and vice versa.

Lemma 6.4. *For all positive integers s, n and every $k \leq n$ and $\varepsilon > 0$ the following holds. Let X be a random variable distributed over $\{0,1\}^n$ for which $H^{\mathrm{metric}^\star}_{s,\varepsilon}(X) < k$. Then, there exists a circuit $D\colon \{0,1\}^n \to [0,1]$ of size $O(s)$ such that $\mathbb{E}[D(U_n)] \leq 2^{k-n}$ and $\mathbb{E}[D(X)] > 2^{n-k}\mathbb{E}[D(U_n)] + \varepsilon$.*

Proof: Let $D\colon \{0,1\}^n \to [0,1]$ be the distinguisher guaranteed by Theorem 6.3, so

$$\mathbb{E}[D(X)] > \mathbb{E}[D(Y^\star)] + \varepsilon.$$

Let $t^\star \in [0,1]$ be the maximal value for which there exists an $x^\star \in \{0,1\}^n$ such that

$$\mathbf{H}_D = \{x \in \{0,1\}^n : D(x) \geq t^\star \wedge x \leq x^\star\}$$

is of size exactly 2^k. Simply put, $\mathbf{H}_D$ contains the first 2^k heaviest elements. Let $Y^\star$ be the uniform distribution over the set $\mathbf{H}_D$ (note that $H_\infty(Y^\star) \geq k$).

We construct $D'\colon \{0,1\}^n \to [0,1]$ as follows. For every $x \in \{0,1\}^n$,

$$D'(x) = \max\{D(x) - t^\star, 0\}.$$

Note that D' is of size $s + O(m) = O(s)$, where m is the number of bits required to represent each element in $\mathrm{Supp}(D)$. Now, observe that

$$\mathbb{E}[D(Y^\star)] - \mathbb{E}[D'(Y^\star)] = \sum_{z \in \mathrm{Supp}(Y^\star)} \Pr[Y^\star = z](D(z) - D'(z)) = \sum_{z \in \mathrm{Supp}(Y^\star)} \Pr[Y^\star = z]t^\star = t^\star,$$

and

$$\mathbb{E}[D(X)] - \mathbb{E}[D'(X)] = \sum_{z \in \mathrm{Supp}(X)} \Pr[X = z](D(z) - D'(z)) \leq \sum_{z \in \mathrm{Supp}(X)} \Pr[X = z]t^\star = t^\star,$$

so

$$\mathbb{E}[D'(X)] - \mathbb{E}[D'(Y^\star)] \geq \mathbb{E}[D(X)] - \mathbb{E}[D(Y^\star)] > \varepsilon.$$

Next, note that $|\mathrm{Supp}(D')| \leq 2^k$, so $\mathbb{E}[D'(U_n)] \leq 2^{k-n}$. Also,

$$\mathbb{E}[D'(U_n)] = \Pr[U_n \in \mathbf{H}_D] \cdot \mathbb{E}[D'(Y^\star)] + \Pr[U_n \notin \mathbf{H}_D] \cdot 0,$$

and thus $\mathbb{E}[D'(Y^\star)] = 2^{n-k}\mathbb{E}[D'(U_n)]$. ■

The alternative characterization of Theorem 6.3 is oblivious to the gates used by the distinguisher. We can thus also state a version of Lemma 6.4 for **FNP**-circuits.

Lemma 6.5. *For every positive integers s, n and every $k \leq n$ and $\varepsilon > 0$ the following holds. Let X be a random variable distributed over $\{0,1\}^n$ for which $H^{\mathbf{N}\text{metric}^\star}_{s,\varepsilon}(X) < k$. Then, there exists an* **FNP***-circuit $D\colon \{0,1\}^n \to [0,1]$ of size $O(s)$ such that $\mathbb{E}[D(U_n)] \leq 2^{k-n}$ and $\mathbb{E}[D(X)] > 2^{n-k}\mathbb{E}[D(U_n)] + \varepsilon$.*

Although we will not use it later, we next prove that the converse also holds and present a more elementary proof that does not use Theorem 6.3.

Lemma 6.6. *For every positive integers s, n and every $k \leq n$ and $\varepsilon > 0$ the following holds. Let X be a random variable distributed over $\{0,1\}^n$ for which $H^{\text{metric}^\star}_{s,\varepsilon}(X) \geq k$. Then, for every circuit $D\colon \{0,1\}^n \to [0,1]$ of size s, it holds that*

$$\mathbb{E}[D(X)] \leq \mathbb{E}[D(U_n)]2^{n-k} + \varepsilon.$$

The same holds for **N***metric** *pseudoentropy and* **FNP***-circuits.*

Before proving the lemma, we will need a corresponding statement for high min-entropy random variables.

Claim 6.7. *Let Y be a random variable over $\{0,1\}^n$ such that $H_\infty(Y) \geq k$, and let $f\colon \{0,1\}^n \to [0,1]$ be any function. Then, $\mathbb{E}[f(Y)] \leq \mathbb{E}[f(U_n)]2^{n-k}$.*

Proof:

$$\mathbb{E}[f(Y)] = \sum_{y\in\{0,1\}^n} \Pr[Y = y]f(y) \leq \sum_{y\in\{0,1\}^n} 2^{-k}f(y) = 2^{n-k}\mathbb{E}[f(U_n)].$$

■

Proof of Lemma 6.6: Suppose towards a contradiction that there exists a circuit $D\colon \{0,1\}^n \to [0,1]$ of size s such that $\mathbb{E}[D(X)] > \mathbb{E}[D(U_n)]2^{n-k} + \varepsilon$. For every $Y \sim \{0,1\}^n$ with $H_\infty(Y) \geq k$ we know, by Claim 6.7, that

$$\mathbb{E}[D(Y)] \leq \mathbb{E}[D(U_n)]2^{n-k}.$$

Thus, $\mathbb{E}[D(X)] - \mathbb{E}[D(Y)] > \varepsilon$. ■

6.2 Extracting Randomness From Metric* Pseudoentropy

Using the min-max theorem, [BSW03] show that for $X \sim \{0,1\}^n$ with $H^{\text{metric}^\star}_{s,\varepsilon}(X) \geq k$, it holds that $H^{\text{HILL}}_{s',\varepsilon}(X) \geq k$, albeit with $s' = O(\frac{s\varepsilon^2}{n})$. The factor n loss in s' is too costly for us. We show that under our complexity-theoretic assumptions, by allowing **FP** gates we are able to directly apply seeded extractors on random variables having high metric pseudoentropy and overcome the loss in circuit size.

The main hurdle we wish to overcome by using **FP** gates is approximating the fraction of accepting inputs to a circuit C to within an constant additive error of ε. We note that the original Nisan-Wigderson generator fools circuits using a number of seeds polynomial in the size of the circuit. Hence taking a sample average over all seeds of the generator yields a polynomial time deterministic approximation of $\mathbb{E}[C(U)]$. To utilize this fact, we define the function problem $\mathrm{DensityApprox}$ as follows. An algorithm solving $\mathrm{DensityApprox}$ gets as input a circuit C and a number $\varepsilon > 0$ and computes the canonical additive ε-approximation to $\mathbb{E}[C(U)]$ rounded to $\log\frac{1}{\varepsilon}$ bits. In light of Theorem 1.1 and the above discussion, we can record the following.

Claim 6.8. *Assume there exists a function in* $\mathbf{E}$ *that requires exponential-size circuits. Then we have that* $\mathrm{DensityApprox} \in \mathbf{FP}$.

We are now ready to prove that one can extract pseudorandomness from $\mathbf{P}$metric* pseudoentropy.

Theorem 6.9. *Assume there exists a function in* $\mathbf{E}$ *that requires exponential-size circuits. Then, for all positive integers* s, t, n, k, d, m *and a constant* $\varepsilon > 0$ *the following holds.*

Suppose $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a* (k, ε) *extractor so that for every* $x \in \{0,1\}^n$, $\mathsf{Ext}(x, \cdot)$ *is computable by a circuit of size* t*. Let* X *be a random variable distributed over* $\{0,1\}^n$ *so that* $H^{\mathbf{P}\mathrm{metric}^\star}_{s',\varepsilon}(X) \geq k$ *for* $s' = 2(s+t)$*. Then,* $\mathsf{Ext}(X, U_d)$ 3ε*-fools circuits of size* s.

Proof: The following lemma will be the crux of the proof.

Lemma 6.10. *For every distinguisher* $C\colon \{0,1\}^m \to \{0,1\}$ *computable by a circuit of size* s *there exists a random variable* $Z \sim \{0,1\}^n$ *with* $H_\infty(Z) \geq k$ *such that*

$$|\mathbb{E}[C(\mathsf{Ext}(X, U_d))] - \mathbb{E}[C(\mathsf{Ext}(Z, U_d))]| \leq 2\varepsilon.$$

Proof: Given any $C\colon \{0,1\}^m \to \{0,1\}$, we define $D_C\colon \{0,1\}^n \to [0,1]$ by

$$D_C(x) = \mathbb{E}_{y \sim U_d}[C(\mathsf{Ext}(x, y))].$$

Assume towards a contradiction that there exists $C\colon \{0,1\}^m \to \{0,1\}$ computable by a circuit of size s for which

$$|\mathbb{E}[C(\mathsf{Ext}(X, U_d))] - \mathbb{E}[C(\mathsf{Ext}(Z, U_d))]| > 2\varepsilon$$

for every $Z \sim \{0,1\}^n$ with $H_\infty(Z) \geq k$. Notice that $\mathbb{E}[D_C(X)] = \mathbb{E}[C(\mathsf{Ext}(X, U_d))]$ and likewise for $\mathbb{E}[D_C(Z)]$. Thus, we have that $|\mathbb{E}[D_C(X)] - \mathbb{E}[D_C(Z)]| > 2\varepsilon$.

Next, we want to approximate D_C by a small $\mathbf{FP}$-circuit. Define $C_x = C(\mathsf{Ext}(x, \cdot))$, and let $D'_C\colon \{0,1\}^n \to [0,1]$ be the circuit using a DensityApprox gate, such that on input x:

- Compute the description of the circuit C_x, having size $s + t$.
- Uses a DensityApprox gate on input $(C_x, \frac{\varepsilon}{4})$ to compute an $\frac{\varepsilon}{4}$-approximation to $\mathbb{E}[C_x(U)]$ rounded to $\log \frac{4}{\varepsilon}$ bits.

The circuit D'_C is of size $s + t + 2\log\frac{4}{\varepsilon} + O(1) \leq 2(s+t)$. Moreover, for every $x \in \{0,1\}^n$, we have an $\frac{\varepsilon}{4}$-approximation to C_x rounded to the first $\log \frac{4}{\varepsilon}$ bits. Hence we have

$$D_C(x) - \frac{\varepsilon}{2} \leq D'_C(x) \leq D_C(x) + \frac{\varepsilon}{2}$$

and so $|\mathbb{E}[D_C(X)] - \mathbb{E}[D'_C(X)]| \leq \frac{\varepsilon}{2}$ and $|\mathbb{E}[D_C(Z)] - \mathbb{E}[D'_C(Z)]| \leq \frac{\varepsilon}{2}$. Recalling that $|\mathbb{E}[D_C(X)] - \mathbb{E}[D_C(Z)]| > 2\varepsilon$, by the triangle inequality we get

$$|\mathbb{E}[D'_C(X)] - \mathbb{E}[D'_C(Z)]| > 2\varepsilon - \varepsilon = \varepsilon.$$

However, the above contradicts the fact that $H^{\mathbf{P}\mathrm{metric}^\star}_{s',\varepsilon}(X) \geq k$. ■

Now, let $C\colon \{0,1\}^m \to \{0,1\}$ be any distinguisher. By the above lemma, there exists $Z \sim \{0,1\}^n$ with $H_\infty(Z) \geq k$ such that

$$|\mathbb{E}[C(\mathsf{Ext}(X, U_d))] - \mathbb{E}[C(\mathsf{Ext}(Z, U_d))]| \leq 2\varepsilon.$$

But we also know, by the extractor property, that $\mathsf{Ext}(Z, U_d) \approx_\varepsilon U_m$, so

$$|\mathbb{E}[C(\mathsf{Ext}(X, U_d))] - \mathbb{E}[C(U_m)]| \leq 3\varepsilon,$$

which proves the lemma. ■

6.3 Yao* Pseudoentropy

We now extend the *compression-based* pseudoentropy notions to handle real-valued distinguishers. This time, we allow for a *lossy* compression scheme, that disregards outputs below a certain threshold.

Definition 6.11. *For some positive integers n, s and $D\colon \{0,1\}^n \to [0,1]$, we say that $D \in \mathcal{C}^{\mathrm{Yao}^\star}_{\ell,s,\eta}$ if there exist circuits $c\colon \{0,1\}^n \to \{0,1\}^\ell$ and $d\colon \{0,1\}^\ell \to \{0,1\}^n \times [0,1]$, each of size s, such that for every $x \in \{0,1\}^n$ with $D(x) \geq \eta$ it holds that $d(c(x)) = (x, D(x))$. We refer to c as the compressing circuit and to d as the decompressing one. Moreover, we say $D \in \mathcal{C}^{\mathbf{N}\mathrm{Yao}^\star}_{\ell,s,\eta}$ if c and d are allowed to be $\mathbf{FNP}$ circuits.*

Note that when D maps to $\{0,1\}$, the above definition coincides with $\mathcal{C}^{\mathrm{Yao}}_{\ell,s}$ for every $\eta < 1$ (see Definition 2.29).

Definition 6.12 (Yao* pseudoentropy)**.** *Let X be a random variable distributed over $\{0,1\}^n$, a positive integer s and $\varepsilon, \eta > 0$. We say that $H^{\mathrm{Yao}^\star}_{s,\varepsilon,\eta}(X) \geq k$ if for every $\ell < k$ and $D \in \mathcal{C}^{\mathrm{Yao}^\star}_{\ell,s,\eta}$,*

$$\mathbb{E}[D(X)] \leq \frac{\mathbb{E}[D(U_n)]2^n}{|\{x \in \{0,1\}^n : D(x) \geq \eta\}|} \cdot 2^{\ell-k} + \varepsilon.$$

We say that $H^{\mathbf{N}\mathrm{Yao}^\star}_{s,\varepsilon,\eta}(X) \geq k$ if we allow $D \in \mathcal{C}^{\mathbf{N}\mathrm{Yao}^\star}_{\ell,s,\eta}$.

As one can hope, the above formulation allows us to relate Yao* pseudoentropy to metric* pseudoentropy.

Lemma 6.13. *For every positive integer n there exists $s_0 = \widetilde{O}(n)$ such that for every positive integer $s \geq s_0$ and $\varepsilon, \eta > 0$ the following holds. Let X be a random variable distributed over $\{0,1\}^n$ such that $H^{\mathbf{N}\mathrm{Yao}^\star}_{s,\varepsilon,\eta}(X) \geq k$. Then, $H^{\mathbf{N}\mathrm{metric}^\star}_{\gamma s,\varepsilon}(X) \geq \frac{k}{2} - \frac{1}{2}\log\frac{1}{\eta}$ where $\gamma < 1$ is some universal constant.*

Proof: Assume towards a contradiction that $H^{\mathbf{N}\mathrm{metric}^\star}_{\gamma s,\varepsilon}(X) < \frac{k}{2} - \frac{1}{2}\log\frac{1}{\eta}$ for a constant $\gamma < 1$ to be determined later on. By Lemma 6.5 there exist a constant $c_1 \geq 1$ and an $\mathbf{FNP}$-circuit $D\colon \{0,1\}^n \to [0,1]$ of size $c_1\gamma \cdot s$ satisfying $\mathbb{E}[D(U_n)] \leq 2^{-n+\frac{k}{2}-\frac{1}{2}\log\frac{1}{\eta}}$ for which

$$\mathbb{E}[D(X)] > \mathbb{E}[D(U_n)]2^{n-\frac{k}{2}+\frac{1}{2}\log\frac{1}{\eta}} + \varepsilon.$$

Denote

$$A = \{x \in \{0,1\}^n : D(x) \geq \eta\},$$

and observe that $|A| < \frac{1}{\eta}2^{\frac{k}{2}-\frac{1}{2}\log\frac{1}{\eta}}$. Set $t = \log|A| < \frac{k}{2} + \frac{1}{2}\log\frac{1}{\eta}$. Let $\mathcal{H} \subseteq \{0,1\}^n \to \{0,1\}^{2t}$ be the two-universal family of hash functions described in Lemma 2.30, of cardinality 2^{2n}. Following Lemma 2.30, we know there exists $h^\star \in \mathcal{H}$, computable by a circuit of size $s_0 = \widetilde{O}(n)$ that is one to one on A.

Set $\ell = 2t$. For a compressing circuit, $c\colon \{0,1\}^n \to \{0,1\}^\ell$ computes $h^\star$. Towards constructing a decompressing $\mathbf{FNP}$-circuit $d\colon \{0,1\}^\ell \to \{0,1\}^n \times [0,1]$, for $i \in [n]$ consider the function problem $\mathrm{Decompress}$ which on input $(z, D, h^\star, \eta)$ returns an x for which $h^\star(x) = z$ and $D(x) \geq \eta$, where we represent $h^\star \in \{0,1\}^{2n} \equiv \mathrm{GF}(2^n)^2$. Clearly, $L \in \mathbf{FNP}$ so we can recover the desired $x \in \{0,1\}^n$.

Thus, d on input $z \in \{0,1\}^\ell$ will use z, the encoding of the hardwired D and hardwired encoding of the good hash function $h^\star$, and a hardwired encoding of η to output x together with $D(x)$. Note that both c and d are of size $c_2c_1\gamma \cdot s$ for some constant $c_2 \geq 1$. Finally, note that

$$\mathbb{E}[D(X)] > \mathbb{E}[D(U_n)]2^{n-\frac{k}{2}+\frac{1}{2}\log\frac{1}{\eta}} + \varepsilon > \mathbb{E}[D(U_n)]2^{n-t+\ell-k} + \varepsilon,$$

which contradicts the fact that $H^{\mathbf{N}\mathrm{Yao}^\star}_{s,\varepsilon,\eta}(X) \geq k$ if we set $\gamma = \frac{1}{c_1c_2}$. ■

7 A Metric* Pseudoentropy Generator From Worst-Case Hardness

In this section we show how to construct an **N**metric* pseudoentropy generator from a function f that is hard for **FNP** circuits. In fact, the same generator from Section 4 used in the SVN case works. Again, let $f\colon \{0,1\}^{\log n} \to \{0,1\}$ be such that every **FNP**-circuit computing f has size at least $n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. That is, $\mathsf{size}_{\mathbf{FNP}}(f) \geq n^{1-\alpha_0}$. Again, we consider the generator $G^f\colon [m] \to \Sigma \equiv \{0,1\}^{\log|\Sigma|}$ defined as

$$G^f(z) = \mathcal{C}(f)_z$$

for a locally list recoverable code

$$\mathcal{C}\colon \{0,1\}^n \to \Sigma^m.$$

Theorem 7.1. *Keeping the above notation, let ε, α be constants such that $\varepsilon > 0$ and $\alpha_0 < \alpha \leq \frac{1}{6}$. Assume $\mathcal{C}$ is $\left(Q, (1-\frac{1}{m})\varepsilon, \ell, s_{\mathcal{C}}\right)$ locally list recoverable so that $s_{\mathcal{C}} = O(n^{1-\alpha})$, and $\frac{n^{1-\alpha}}{Q} \geq s_0$ for some $s_0 = \widetilde{O}(\log|\Sigma|)$. Then, G^f is a (k, s, ε) **N**metric* PEG for $k = \frac{\log \ell}{2} - \frac{1}{2}\log\frac{m}{\varepsilon}$ and $s = \Omega\left(\frac{n^{1-\alpha}}{Q}\right)$.*

Proof: Let Z be the uniform distribution over $[m]$. We first show that

$$H^{\mathbf{N}\mathrm{Yao}^\star}_{s',\varepsilon,\eta}\left(Z \circ G^f(Z)\right) \geq k'$$

for $k' = \log \ell$, $s' = \frac{n^{1-\alpha}}{Q}$ and $\eta = \frac{\varepsilon}{m}$.

Assume towards a contradiction that $H^{\mathbf{N}\mathrm{Yao}^\star}_{s',\varepsilon,\eta}(Z \circ G^f(Z)) < k'$. Then there must exist a $D\colon [m] \times \Sigma \to [0,1]$, with $D \in C^{\mathbf{N}\mathrm{Yao}^\star}_{k',\varepsilon,\eta}$ such that

$$\mathbb{E}[D(Z \circ G^f(Z))] > \varepsilon$$

By an averaging argument, this means that for at most $1 - \varepsilon(1-\frac{1}{m})$ fraction of $z \in [m]$, we have $D(z, C(f)_z) < \frac{\varepsilon}{m}$. Since $D \in C^{\mathbf{N}\mathrm{Yao}^\star}_{k',\varepsilon,\eta}$, there exist compressing and decompressing **FNP** circuits $c\colon [m] \times \Sigma \to \{0,1\}^{k'}$ and $d\colon \{0,1\}^{k'} \to [m] \times \Sigma \times [0,1]$ such that for every $(y,\sigma) \in [m] \times \Sigma$ with $D(z,\sigma) > \eta = \frac{\varepsilon}{m}$, we have $d(c(z,\sigma)) = ((z,\sigma), D(z,\sigma))$.

Consider sets $S_1, \ldots, S_m \subseteq \Sigma$ defined as follows:

$$S_z = \{\sigma \in \Sigma : \exists y \in \{0,1\}^{k'}, \gamma \in [0,1] \text{ s.t. } d(y) = ((z,\sigma),\gamma)\}.$$

Notice that by definition, $\sum_{z\in[m]} |S_z| = 2^{k'} = \ell$. Moreover, for at least $\varepsilon(1-\frac{1}{m})$ of z-s we know that $C(f)_z \in S_z$. Thus by definition of our list recoverable code, there exists a circuit A computing f that is of size at most $s_{\mathcal{C}}$, that uses at most Q oracle queries to the lists S_z. We can replace each oracle gate with a circuit that computes the decompressor circuit on the appropriate string in $\{0,1\}^{k'}$, and outputs the resulting symbol. Since the size of the decompressing circuit is at most s', we have a final circuit of size

$$O(s_{\mathcal{C}} + Q \cdot s') = O(n^{1-\alpha})$$

that computes f, which is a contradiction of the hardness of f.

Knowing that $H^{\mathbf{N}\mathrm{Yao}^\star}_{s,\varepsilon,\eta}\left(Z \circ G^f(Z)\right) \geq k'$, and $s' = \widetilde{\Omega}(\log|\Sigma|)$, applying Lemma 6.13 proves the theorem. ∎

Combining Theorem 3.2 with the above result immediately gives the following corollary.

Corollary 7.2. *For every positive integer n and a constant $\varepsilon > 0$ the following holds. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ is such that $\mathsf{size}_{\mathbf{FNP}}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{6}$. Then, there exists a function*

$$G^f : \{0,1\}^d \to \{0,1\}^m$$

that is a (k, s, ε) $\mathbf{N}$metric* PEG for $k = \frac{n^{1-7\alpha}}{2} - O(\log n)$, $s \geq n^{1-4\alpha}$, $d = 5\alpha \log n + O(\log\log n)$ and $m = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n^2)$ to compute. Moreover, if $f \in \mathbf{DTIME}(n^{c_f})$ for some constant $c_f \geq 1$, the support of G^f can be computed in time $\widetilde{O}(n^{c_f+1})$.

The proof is almost identical to that of Theorem 4.2.

8 Extractors With Near-Optimal Seed Length

We give an explicit strong seeded extractor with near-optimal (in n) seed that supports min-entropy $n^{1-\alpha}$ for every $\alpha < \frac{1}{2}$. Specifically, for $\varepsilon = n^{-o(1)}$, our seed length is $(1 + O(\alpha)) \log n$, which gives a left-degree of $n^{1+O(\alpha)}$. A very similar construction was given in [TZS06], but there the analysis was only done for a constant entropy rate. For our construction we will use three ingredients given in the following theorems.

The first theorem, due to Ta-Shma, Zuckerman and Safra gives an extractor that achieves seed length very close to $\log n$; however, it only outputs a small portion of the min-entropy.

Theorem 8.1 ([TZS06])**.** *For all positive integers n, k, m and $\varepsilon \leq \frac{1}{2}$ such that $3m\sqrt{n} \log \frac{n}{\varepsilon} \leq k \leq n$, there exists an explicit strong (k, ε) extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{\Omega(m)}$ for $d = \log n + O(\log \frac{1}{\varepsilon}) + O(\log m)$.*

The next extractor is Trevisan's, with improved analysis due to Raz, Reingold and Vadhan. We use it to output a constant fraction of the min-entropy.

Theorem 8.2 ([Tre01, RRV02])**.** *There exists a constant $c_{\mathsf{Tre}} \geq 1$ such that following holds. For all positive integers n, k and $\varepsilon > 0$ there exists an explicit strong (k, ε) extractor $\mathsf{Tre}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for $d = c_{\mathsf{Tre}} \log^2 \frac{n}{\varepsilon}$ and $m = \frac{k}{4}$.*

Lastly, we will need the following condenser by Reingold, Shaltiel and Wigderson whose purpose is to output a shorter string than the input source with the same entropy rate, using a very short seed.

Theorem 8.3 ([RSW06])**.** *There exist constants $c_{\mathsf{RSW}} \geq 1$ and $0 < \gamma_{\mathsf{RSW}} < 1$ such that the following holds. For all positive integers n, k, r and $\varepsilon \geq c_{\mathsf{RSW}} \sqrt{\frac{r}{k}}$ there exists an explicit strong $(k, k' = \gamma_{\mathsf{RSW}} \frac{k}{r}, \varepsilon)$ condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for $d = c_{\mathsf{RSW}}(\log\log n + \log r + \log \frac{1}{\varepsilon})$ and $m = \frac{n}{r}$.*

Our construction, following [TZS06], proceeds along a familiar paradigm. The extractor TZS uses a short seed but extracts relatively few bits, whereas the extractor Tre outputs a constant fraction of the min-entropy but requires a long seed. We are able to utilize both advantages without suffering the drawbacks by using RSW, with a very short seed, to condense the source X. The output length of RSW will be only half the min-entropy, enabling us to use TZS to extract (from the same X) a seed for Tre. Namely, our extractor outputs $\mathsf{Ext}(X, Y_1 \circ Y_2) = \mathsf{Tre}(\mathsf{RSW}(X, Y_1), \mathsf{TZS}(X, Y_2))$, where Y_1 and Y_2 are independent uniform seeds. Note that $\mathsf{RSW}(X, Y_1)$ and $\mathsf{TZS}(X, Y_2)$ can be dependent, however we will argue that they are *close* to being independent, and therefore we can apply Tre.

We are now ready for the details. We are given a positive integer n, $\varepsilon > 0$, some constant $\alpha < \frac{1}{2}$ and $k \geq n^{1-\alpha}$. As promised, we will make use of the following ingredients.

- The strong $(k, \gamma_{\mathsf{RSW}}\frac{k}{r}, \varepsilon)$ condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ for $r = 2n^{\alpha}$, where $m_1 = \frac{1}{2}n^{1-\alpha}$ and $d_1 = c_{\mathsf{RSW}}(\log\log n + \log(2n^{\alpha}) + \log\frac{1}{\varepsilon})$. We require $\varepsilon \geq c_{\mathsf{RSW}}\sqrt{\frac{r}{k}} = \Omega\left(\frac{1}{n^{\frac{1}{2}-\alpha}}\right)$.
- The strong (k', ε) extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{d_3}$ where $k' = \frac{1}{4}n^{1-\alpha}$, $d_3 = c_{\mathsf{Tre}}\log^2\frac{m_1}{\varepsilon} = O(\log^2\frac{n}{\varepsilon})$ and $d_2 = \log n + O(\log\frac{1}{\varepsilon}) + O(\log d_3)$. Note that indeed k' is large enough.
- The strong (k'', ε) extractor $\mathsf{Tre}\colon \{0,1\}^{m_1} \times \{0,1\}^{d_3} \to \{0,1\}^m$ for $k'' = \frac{\gamma_{\mathsf{RSW}}}{2}n^{1-2\alpha}$ and $m = \frac{k'}{4} = \frac{\gamma_{\mathsf{RSW}}}{8}n^{1-2\alpha}$.

Given $x \in \{0,1\}^n$, $y_1 \in \{0,1\}^{d_1}$ and $y_2 \in \{0,1\}^{d_2}$, we outputs

$$\mathsf{Ext}(x, y_1 \circ y_2) = \mathsf{Tre}(\mathsf{RSW}(x, y_1), \mathsf{TZS}(x, y_2)).$$

Theorem 8.4. *There exists a constant $c \geq 1$ such that the following holds for any constant $\alpha < \frac{1}{2}$. The function $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ above is an explicit strong (k, ε) extractor for every positive integer n, $k \geq n^{1-\alpha}$ and $\varepsilon \geq cn^{-\frac{1}{2}+\alpha}$, where $d = (1 + c\alpha)\log n + c\log(\frac{1}{\varepsilon})$ and $m = \frac{1}{c}n^{1-2\alpha}$.*

Proof: The proof will mimic block-source extraction techniques, but is self-contained. Denote $A = \mathsf{RSW}(X, Y_1)$ and $B = \mathsf{TZS}(X, Y_2)$. By the properties of TZS, we know that

$$(B, Y_2) \approx_{\varepsilon} U_{d_3} \times Y_2.$$

By the properties of RSW, there exists A', which is ε-close to A, for which $H_{\infty}(Y_1 \circ A') \geq k'' + d_1$ (note that A' depends on Y_1). By Lemma 2.13, and since X and Y_1 are independent,

$$\widetilde{H}_{\infty}(X | Y_1 \circ A') \geq H_{\infty}(X) - m_1 = \frac{1}{2}n^{1-\alpha} \geq \frac{1}{4}n^{1-\alpha} + \log\frac{1}{\varepsilon}.$$

Since X and Y_2 are independent of each other and of Y_1, and since A' is independent of Y_2, we know that X, Y_2 are also independent conditioned on $Y_1 \circ A'$. Applying Lemma 2.15 on the strong extractor TZS we get that

$$(A', B, Y_1, Y_2) \approx_{2\varepsilon} (A', U_{d_3}, Y_1, Y_2).$$

Applying Tre, we deduce that

$$(\mathsf{Tre}(A', B), Y_1, Y_2) \approx_{2\varepsilon} (\mathsf{Tre}(A', U_{d_3}), Y_1, Y_2).$$

Now, $\mathsf{Tre}(A', U_{d_3}) \approx_{\varepsilon} U_m$ so by the triangle inequality,

$$(\mathsf{Tre}(A', B), Y_1, Y_2) \approx_{3\varepsilon} (U_m, Y_1, Y_2).$$

Accounting for the distance between A and A', we finally get that

$$(\mathsf{Ext}(X, Y_1 \circ Y_2), Y) \approx_{4\varepsilon} (U_m, Y),$$

as desired. To conclude, observe that

$$d = d_1 + d_2 = (1 + O(\alpha))\log n + O\left(\log\frac{1}{\varepsilon}\right).$$

The explicitness follows from the explicitness of each component. ■

8.1 Computing the Extractor by a Small Circuit

Usually, when constructing extractors, we require the computation to be done in polynomial time or in linear space. In this work, we must exercise a more fine-grained analysis, since the size it takes to compute Ext corresponds to the minimal circuit size we will be able to fool with a nearly-optimal small slowdown (see Theorem 6.9 for the precise parameters). We prove that there exists a circuit of nearly-linear size that computes Ext. But before doing so, we will need to argue that a few basic pseudorandomness primitives are also computable in nearly-linear size.

8.1.1 Efficient Asymptotically-Good Codes

An asymptotically-good code is a family of binary codes having a constant rate and a constant relative distance. An example is Justesen's code [Jus72].

Lemma 8.5 (following [Jus72])**.** *There exists a constant $0 < \delta < 1$ such that the following holds. For every positive integer n there exists an explicit $[\bar{n}, n, \delta]_2$ code $\mathcal{C}_{\mathsf{Jus}} \colon \{0,1\}^n \to \{0,1\}^{\bar{n}}$ of rate $\frac{n}{\bar{n}} \geq \delta$.*

Moreover, for every positive integer n there exists a circuit C_n of size $\widetilde{O}(n)$ so that for every $x \in \{0,1\}^n$, $C_n(x)$ computes $\mathcal{C}_{\mathsf{Jus}}(x)$.

Proof: The code is constructed as follows. For some $\bar{n} = O(n)$ and $m = O(\log n)$, let $\mathbb{F}_q$ be a finite field of cardinality $q = 2^m$. Set $k = \frac{n}{m}$, $\bar{k} = \frac{\bar{n}}{2m}$ and let $\{\alpha_1, \ldots, \alpha_{\bar{k}}\}$ be some distinct elements of $\mathbb{F}_q^\star$. Given $x \in \{0,1\}^n \equiv \mathbb{F}_1^k$, let p_x be the univariate polynomial $p_x(\alpha) = \sum_{i=0}^{k-1} x_i \alpha^i$. Then, the encoding is given by

$$\mathcal{C}_{\mathsf{Jus}}(x) = ((p_x(\alpha_1), \alpha_1 p_x(\alpha_1)), \ldots, (p_x(\alpha_{\bar{n}}), \alpha_{\bar{n}} p_x(\alpha_{\bar{n}}))) \in \{0,1\}^{\bar{n}},$$

where we interpreted each element of $\mathbb{F}_1$ as a string in $\{0,1\}^m$. The fact that the Justesen's code is asymptotically good (i.e., that one can take such parameters and maintain a constant relative distance) is by now standard [Jus72, GRS19], but we still need to justify that fact that the encoding can be done via a small circuit.

For simplicity, consider hardwiring $\{\alpha_1, \ldots, \alpha_{\bar{k}}\}$, which takes $m\bar{k} + O(1) = O(n)$ bits. Each bit of $\mathcal{C}_{\mathsf{Jus}}(x)$ is obtained by evaluating p_x on some α_i, either multiplying by α_i or not, and then taking a specific coordinate in the resulting element's binary encoding. For the evaluation step, we can use fast univariate multi-point evaluation which can be done by a circuit of size $\bar{k} \cdot \mathrm{poly}(\log q) = \widetilde{O}(n)$ (see, e.g., [vzGG13, Section 10]). All other operations can be done, per output bit, by a circuit of size $\mathrm{poly}(\log q)$, so overall computing $\mathcal{C}_{\mathsf{Jus}}(x)$ can be done by a circuit of size $\widetilde{O}(n)$. ∎

8.1.2 Efficient List-Decodable Codes

Good binary list-decodable codes are implied by explicit construction of small *ε-balanced codes* close to the Gilbert-Varshamov (GV) bound. We will not define these object explicitly, and just say that they give rise to $(\frac{1}{2} - \varepsilon, \varepsilon^{-2})$ list-decodable codes of length $\frac{n}{\varepsilon^c}$ for some constant $c \geq 2$. The constant c, as well as achieving small rate with non-optimal dependence on n, has been subject to an important research. Naor and Naor [NN93] obtain $c = 3$, and Ta-Shma [Ta-17] recently achieved a near-optimal $c = 2 + \alpha$ for every constant $\alpha > 0$. For our construction we use the following code having $c = 4 + \alpha$ for every constant $\alpha > 0$ (for simplicity, we state it for $c = 5$). The construction, based on distance amplification via expander walks, is given in [Ta-17] and was inspired by an unpublished result by Rozenman and Wigderson.

Lemma 8.6 (following [Ta-17])**.** *For every positive integer n and $\varepsilon = \varepsilon(n)$ there exists a binary error correcting code $\mathcal{C}\colon \{0,1\}^n \to \{0,1\}^{\bar{n}}$ which is $(\frac{1}{2} - \varepsilon, \varepsilon^{-2})$ list-decodable, for $\bar{n} = O(\frac{n}{\varepsilon^5})$, computable by a circuit C_n of size $\widetilde{O}(\bar{n})$.*

Proof: Let $\mathcal{C}_{\mathsf{Jus}}\colon \{0,1\}^n \to \{0,1\}^{cn}$ be Justesen's code, given to us in Lemma 8.5, for some constant $c > 1$. Let $G = (V = [cn], E)$ be a Ramanujan λ-expander of degree D, where we take D to be a large enough constant so that λ is half the bias of $\mathcal{C}_{\mathsf{Jus}}$.[17] We think of every vertex $v \in V$ as being labeled with some row of the $cn \times n$ generating matrix A_{Jus} of $\mathcal{C}_{\mathsf{Jus}}$. We refer to that labelling by $l(v) \in \{0,1\}^n$.

We now describe how one obtains each row of the $\bar{n} \times n$ generating matrix A of $\mathcal{C}$. For $t = 5\log_D \frac{1}{\varepsilon}$, each such row is the sum, modulo 2, of the codewords that appear along a length-t walk over G. That is, each row out of the $\bar{n} = n \cdot D^t = \frac{n}{\varepsilon^5}$ possible rows is indexed by some path $p_i = v_0^i \sim \ldots \sim v_t^i$ in G and its value is

$$\sum_{j=0}^{t} l(v_j^i) \in \{0,1\}^n .$$

Thus, given $x \in \{0,1\}^n$, computing $\mathcal{C}(x)$ is done by outputting

$$Ax = \begin{pmatrix} P_1^\dagger \\ \vdots \\ P_{\bar{n}}^\dagger \end{pmatrix} \cdot A_{\mathsf{Jus}} \cdot x,$$

where P_i is the vector of length cn in which $P_i(v) = 1$ wherever $v \in p_i$ and 0 elsewhere.

For simplicity, consider hardwiring the encoding of each of the $\bar{n}$ possible paths. That is, for every $i \in [\bar{n}]$ we keep a string of length $(t+1)\log(cn)$ storing the vertices along p_i. This hardwiring takes $\widetilde{O}(\bar{n})$ bits overall. Now, given $x \in \{0,1\}^n$, computing $A_{\mathsf{Jus}} \cdot x = \mathcal{C}_{\mathsf{Jus}}(x)$ is done once, which by Lemma 8.5 can be computed with a circuit of size $\widetilde{O}(n)$. To get $\mathcal{C}(x)$ from $\mathcal{C}_{\mathsf{Jus}}(x)$ we need to sum, for each $i \in [\bar{n}]$, the $t+1$ coordinates in $\mathcal{C}_{\mathsf{Jus}}(x)$ that corresponds to the path p_i. As we have those coordinates hardwired, it takes only $\widetilde{O}(t\bar{n}) = \widetilde{O}(\bar{n})$ size, and overall the lemma follows. ■

We note that if we insist on a *uniform* computation of $\mathcal{C}$ we can still work with good enough explicit expanders and *compute* the vectors p_i instead of fixing them (see Theorem 2.20). This would take $\widetilde{O}(\frac{n}{\varepsilon^c})$ time and possibly deteriorate the constant c by a bit.

8.1.3 Almost k-wise Independent Sample Spaces

Definition 8.7 (almost k-wise distribution)**.** *A distribution $(X_1, \ldots, X_n)$ over $\{0,1\}^n$ is (k, ε, p) independent if for every subset $\{i_1, \ldots, i_k\} \subseteq [n]$, $(X_{i_1}, \ldots, X_{i_k})$ is ε-close to the distribution over k-bit strings where all bits are independent and each of them takes the value 1 with probability p.*

Constructing almost k-wise distribution with optimal support size, at least for the $p = \frac{1}{2}$ case, is well-known. In what follows, we argue that sampling from the support of such a distribution can be done by a small circuit. For simplicity, we restrict ourselves to the case of $k = 2$.

[17] By Ramanujan here we mean that λ is optimally related to the degree D, i.e., $\lambda \approx \frac{2}{\sqrt{D}}$. We do not insist on having $\lambda \leq \frac{2\sqrt{D-1}}{D}$ and do allow some slackness, and such graphs exist for every n [F+03].

Lemma 8.8. *For every positive integer n, $\varepsilon = \varepsilon(n)$ and $q = q(n)$ the following holds. Set $\ell = \frac{4q^2 \log^2(qn)}{\varepsilon^2}$. Then, there exists a circuit $C_n \colon [\ell] \to \{0,1\}^n$ of size $\widetilde{O}(\frac{nq}{\varepsilon^2})$ such that for every $y \in [\ell]$, $C_n(y)$ computes the y-th element of a $(2, \varepsilon, 2^{-q})$ independent sample space in some fixed order.*

Proof: Constructing almost k-wise samples spaces for $p = \frac{1}{2}$ is done by combining truly k-wise independence and small-bias sample spaces. Following, e.g., [RSW06], we unfold the extension to a general $p = 2^{-q}$ and argue that it can be done efficiently. Set $h = 2q\log(qn)$, and so $\ell = (\frac{h}{\varepsilon})^2$. We use the following two ingredients:

- Let $B \subseteq \{0,1\}^h$ be an ε-biased sample space.[18] Alon et al. [AGHP92] gives us a simple construction with support size ℓ.
- Let A be the $nq \times h$ generator matrix of a $(2q, 0, \frac{1}{2})$ independent sample space. The matrix A is some suitable Vandermonde matrix, converted to $\mathbb{F}_2$ in a canonical way.

Then, given $y \in [\ell]$, we output the y-th element of the almost k-wise sample space as follows.

1. Let $x \in \{0,1\}^h$ be the y-th element of B. As B is small (and of course, independent of y) we can hardwire B into our circuit, and it takes only $O(\ell h)$ bits.
2. Compute $w = Ax \in \{0,1\}^{nq}$. The matrix-vector multiplication can be done efficiently, exploiting the special structure of A. Specifically, one can use discrete Fourier transform to compute w by a circuit of size $\widetilde{O}(nq)$.
3. Partition w into n consecutive blocks $W_1, \ldots, W_n$, each of length q. Output $z \in \{0,1\}^n$ such that $z_i = 1$ if and only if all the bits in W_i are 1. This step can be implemented by a circuit of size $O(nq)$.

Overall, the three steps above can be implemented by a circuit of size $\widetilde{O}(nq) + O(\ell h) = \widetilde{O}(\frac{nq}{\varepsilon^2})$, as required. ■

Again, a note about uniform computation is in order. Here too we have an algorithm running in time $\widetilde{O}(n) \cdot \mathrm{poly}(\frac{q}{\varepsilon})$, by explicitly computing the y-th element of B. We skip the details.

8.1.4 Efficiently Computing Ext

Finally, we are ready to prove that Ext can be computed by a small circuit.

Lemma 8.9. *There exists a constant $c \geq 1$ such that the following holds. For some constant $0 < \alpha < \frac{1}{2}$, positive integer n and $\varepsilon > 0$, let $\mathsf{Ext} \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be the $(k = n^{1-\alpha}, \varepsilon)$ strong extractor given in Theorem 8.4. For any fixed $x \in \{0,1\}^n$, the function*

$$\mathsf{Ext}(x, \cdot) \colon \{0,1\}^d \to \{0,1\}^m$$

can be computed by a circuit C_x of size $\widetilde{O}(\frac{n}{\varepsilon^c})$, and it takes a circuit of size $\widetilde{O}(\frac{n}{\varepsilon^c})$ to compute the encoding of C_x.

Proof: We keep the same notation as in our construction.

[18] Namely, for any nonempty $S \subseteq [h]$, we require that $\left|\Pr_{b \sim B}[\bigoplus_{i \in S} b_i = 1] - \frac{1}{2}\right| \leq \varepsilon$.

- The extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{d_3}$ from Theorem 8.1 is computed as follows. For $x \in \{0,1\}^n$ and $y_2 \in \{0,1\}^{d_2}$, we view x as a bivariate polynomial $f_x\colon \mathbb{F}^2 \to \mathbb{F}$ of total degree $h \approx \sqrt{n}$ where $|\mathbb{F}| = h \cdot \mathrm{poly}(d_3/\varepsilon)$. Then, for each $i \in [d_3]$,
$$\mathsf{TZS}(x, y_2)_i = \mathcal{C}(f_x(a_1 + i, a_2))_j,$$
where y_2 is interpreted as $(a_1, a_2, j) \in \mathbb{F}^2 \times [\ell]$ and $\mathcal{C}\colon \mathbb{F} \to [\ell]$ is $(\frac{1}{2} - \rho, \rho^{-1})$ list decodable for $\rho = \Omega(\varepsilon^2/d_3^2)$. Given $x \in \{0,1\}^n$, evaluating f_x on d_3 inputs can be naively done by a circuit of size $d_3 \cdot h^2 \cdot \mathrm{polylog}(|\mathbb{F}|) = \widetilde{O}(n \log \frac{1}{\varepsilon})$. Applying $\mathcal{C}$, by Lemma 8.6 takes size $\widetilde{O}(\frac{n}{\rho^5}) = \widetilde{O}(\frac{n}{\varepsilon^{10}})$. Thus, $\widetilde{O}(\frac{n}{\varepsilon^{10}})$ is also the size it takes to *prepare* the encoding of the circuit that computes $\mathsf{TZS}(x, \cdot)$.

- The condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ from Theorem 8.3 can be computed as follows. Let $\mathcal{C}_{\mathsf{Jus}}\colon \{0,1\}^n \to \{0,1\}^{\bar{n}=O(n)}$ be Justesen's code from Lemma 8.5. Let
$$\left\{ S_y \in \{0,1\}^{\bar{n}} : y \in \{0,1\}^{d_1} \right\}$$
be a $(2, \varepsilon, p)$ independent sample space for $p = \Theta(\frac{m_1}{\bar{n}})$, where we identify each S_y as a subset of $[\bar{n}]$. We also require that $|S_y| = pn$ for every y. Although it is not guaranteed by the construction of Lemma 8.8, [RSW06] show that enforcing this constraint by adding or removing arbitrary indices from each set is good enough. For $x \in \{0,1\}^n$ and $y_1 \in \{0,1\}^{d_1}$, the construction is given by
$$\mathsf{RSW}(x, y_1) = \mathcal{C}(x)_{S_{y_1}}.$$
Computing the encoding of a circuit that computes $\mathsf{RSW}(x, \cdot)$ starts by computing $\mathcal{C}(x)$, which can be done in size $\widetilde{O}(n)$ using Lemma 8.5. On input y_1, computing $\mathcal{C}(x)_{S_{y_1}}$ can be implemented in size $\widetilde{O}(\frac{\bar{n}\log(1/p)}{\varepsilon^2}) = \widetilde{O}(\frac{n}{\varepsilon^2})$, by Lemma 8.8. Thus, it takes $\widetilde{O}(\frac{n}{\varepsilon^2})$ size to compute the encoding of the circuit $\mathsf{RSW}(x, \cdot)$, which is also an upper bound on its size.

- The extractor $\mathsf{Tre}\colon \{0,1\}^{m_1} \times \{0,1\}^{d_3} \to \{0,1\}^m$ goes as follows. Let $\mathcal{C}\colon \{0,1\}^{m_1} \to \{0,1\}^{\overline{m_1}}$ be a $(\frac{1}{2} - \rho, \rho^{-2})$ list decodable code, for $\rho = \Omega(\varepsilon/m)$. Let
$$\{S_i \subseteq [d_3] : i \in [m]\}$$
be a weak design, wherein each $|S_i| = \log(\overline{m_1})$ and for all $i \neq j$, $\sum_{j<i} 2^{|S_i \cap S_j|} \le 2m$. Then, given $x \in \{0,1\}^{m_1}$ and $y \in \{0,1\}^{d_3}$, for each $i \in [m]$ we have that
$$\mathsf{Tre}(x, y)_i = \hat{x}(y|_{S_i}),$$
where we denoted $\hat{x} = \mathcal{C}(x)$. Given $x \in \{0,1\}^{m_1}$ and $y \in \{0,1\}^{d_3}$, a circuit outputting $\mathsf{Tre}(x, y)$ can be constructed as follows. The sets $S_1, \ldots, S_m$ can be hard-coded to the circuit, which takes $m \cdot d_3 = O(n)$ bits. As $\overline{m_1}$ is large, we should not compute $\hat{x}$ in full, but rather compute $y|_{S_1}, \ldots, y|_{S_m}$ at first, and then proceed to computing $\hat{x}(y|_{S_1}), \ldots, \hat{x}(y|_{S_m})$.

  Computing $y|_{S_1}, \ldots, y|_{S_m}$ is immediate once we have $S_1, \ldots, S_m$ and can be done in size $O(md_3) = O(n)$. For computing $\hat{x}(y|_{S_1}), \ldots, \hat{x}(y|_{S_m})$, we revisit the proof of Lemma 8.6. We see that we can first compute $\mathcal{C}_{\mathsf{Jus}}(x)$, by a circuit of size $\widetilde{O}(m_1) = O(n)$, and then for every $z = y|_{S_i} \in \{0,1\}^{\log(\overline{m_1})}$, we interpret z as a walk p_z of length $t = O(\log \frac{m}{\varepsilon})$ over an expander with fully-explicit neighbourhood function (see Theorem 2.20). This takes size $\mathrm{polylog}(n, \frac{1}{\varepsilon})$. From p_z we can compute $\hat{x}(z)$ as described in the proof of Lemma 8.6. We conclude that a circuit of size $O(n)$ suffices to compute Tre.

Overall, accumulating the sizes, the lemma follows. ■

Our extractor is also time-efficient.

Lemma 8.10. *For some constant $0 < \alpha < \frac{1}{2}$, positive integer n and a constant $\varepsilon > 0$, let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be the $(k = n^{1-\alpha}, \varepsilon)$ strong extractor given in Theorem 8.4. Then, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, $\mathsf{Ext}(x,y)$ is computable in time $\widetilde{O}_\varepsilon(n)$.*

We skip the details, and just note that to derive Lemma 8.10 from the above discussion, it is left to verify that computing the weak design can be done efficiently. Indeed, inspecting the construction of [RRV02], this can be done in time $m \cdot \mathrm{polylog}(n) = O(n)$.

9 PRGs With Nearly Optimal Slowdown

All ingredients are now in place for our PRG transforming almost all the hardness to pseudorandom bits using a nearly optimal seed length.

Theorem 9.1. *There exists a constant $c \geq 7$ such that the following holds for every positive integer n and a constant $\varepsilon > 0$. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ with $\mathsf{size}_{\mathbf{FNP}}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{c}$. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{c}$. Then, there exists a function*

$$\overline{G}^f\colon \{0,1\}^{(1+c\alpha)\log s} \to \{0,1\}^s$$

which is an ε-PRG against circuits of size

$$s = n^{1-c\alpha}.$$

The support of $\overline{G}^f$ can be computed in time $s^{\frac{2+c\alpha}{1-c\alpha}}$ given oracle access to the truth table of f. Moreover, if $f \in \mathbf{DTIME}(s^{c_f})$ for some $c_f \geq 1$ then the support of $\overline{G}^f$ can be computed in time $s^{\frac{\gamma}{1-c\alpha}}$ for $\gamma = \max\{2 + c\alpha, c_f + 1\}$.

Proof: Set $\varepsilon' = \frac{\varepsilon}{3}$. Let

$$G^f\colon \{0,1\}^{d_1 = 6\alpha\left(1+\frac{7}{2}\alpha\right)\log s'} \to \{0,1\}^{m_1}$$

be the (k, s', ε') **Nmetric*** PEG guaranteed by Corollary 7.2, where $s' = n^{1-\frac{7}{2}\alpha}$, $m_1 = n^{1-4\alpha}$ and $k = n^{1-8\alpha}$. Let

$$\mathsf{Ext}\colon \{0,1\}^{m_1} \times \{0,1\}^{d_2} \to \{0,1\}^m$$

be the (k, ε') extractor guaranteed to us by Theorem 8.4, so that for every $x \in \{0,1\}^{m_1}$, $\mathsf{Ext}(x, \cdot)$ is computable by a circuit of size $t = \widetilde{O}_\varepsilon(m_1) = o(s')$. By Theorem 8.4, $d_2 = (1 + c_2\alpha)\log m_1 + O_\varepsilon(1)$ and $m = m_1^{1-c_2\alpha}$ for some universal constant $c_2 \geq 1$. Let

$$\overline{G}^f\colon \{0,1\}^{d = d_1 + d_2} \to \{0,1\}^m$$

be such that for $(y_1, y_2) \in \{0,1\}^{d_1} \times \{0,1\}^{d_2}$,

$$\overline{G}^f(y_1, y_2) = \mathsf{Ext}(G_f(y_1), y_2).$$

Denote $X = G^f(U_{d_1})$. By the properties of the PEG, $H^{\mathbf{N}\text{metric}^\star}_{s',\varepsilon'}(X) \geq k$. This also means that $H^{\mathbf{P}\text{metric}^\star}_{s',\varepsilon'}(X) \geq k$, and so by Theorem 6.9, there exists a constant $c' \geq 1$ such that $\mathsf{Ext}(X, U_{d_2})$ $3\varepsilon'$-fools circuits of size $s = \frac{1}{c'}s' - t \geq \frac{1}{2c'}s'$. Note that $m = n^{(1-4\alpha)(1-c_2\alpha)} \leq s$ and

$$\begin{aligned}
d &= d_1 + d_2 \\
&= 6\alpha\left(1 + \frac{7}{2}\alpha\right)\log s' + (1 + c_2\alpha)\log m_1 + O_\varepsilon(1) \\
&\leq 6\alpha\left(1 + \frac{7}{2}\alpha\right)\log s + (1 + c_2\alpha)\left(1 - \frac{7}{2}\alpha\right)(1-4\alpha)(1+7\alpha)\log s + O_\varepsilon(1) \\
&\leq \left(6\alpha + \frac{6\cdot 7}{2}\alpha^2\right)\log s + (1 + 3\alpha + 4\alpha c_2)\log s + O_\varepsilon(1) \\
&\leq 10\alpha\log s + (1 + 3\alpha + 4\alpha c_2)\log s \\
&\leq (1 + 13\alpha + 4\alpha c_2)\log s,
\end{aligned}$$

where we use the fact that $\alpha \leq \frac{1}{7}$. The first part of the theorem then holds with $c = 4c_2 + 13$.

Finally, we address the time it takes to compute the support of $\overline{G}^f$. If we are given oracle access to f, it requires $\widetilde{O}(n^2) = \widetilde{O}(s^{\frac{2}{1-c\alpha}})$ time to compute the support of the PEG G^f. If we assume $f \in \mathbf{DTIME}(s^{c_f})$, then it takes time $n^{c_f+1} = s^{\frac{c_f+1}{1-c\alpha}}$ to compute f at every input and so overall it takes time

$$\widetilde{O}\left(s^{\frac{c_f+1}{1-c\alpha}}\right)$$

to compute the support of the PEG. By Lemma 8.10, the extractor takes time $\widetilde{O}_\varepsilon(m_1)$ on a single seed. To compute the support of $\overline{G}^f$, we compute the support of the PEG, and for every element of the support, we run the extractor on every seed. This takes time

$$\widetilde{O}(n^{5\alpha}) \cdot 2^{d_2} \cdot \widetilde{O}_\varepsilon(m_1) = \widetilde{O}_\varepsilon\left(n^{5\alpha} m_1^{2+c_2\alpha}\right) = \widetilde{O}_\varepsilon\left(n^{5\alpha+(1-4\alpha)(2+c_2\alpha)}\right) \leq s^{\frac{2+c\alpha}{1-c\alpha}}.$$

Thus overall, if we have oracle access to f, computing the support of $\overline{G}^f$ takes time $s^{\frac{2+c\alpha}{1-c\alpha}}$. If we assume $f \in \mathbf{DTIME}(s^{c_f})$, then it takes time $s^{\frac{\gamma}{1-c\alpha}}$ for $\gamma = \max\{2 + c\alpha, c_f + 1\}$. ■

Remark 9.2. *Inspecting the components of the proof of Theorem 9.1, one can see that we assume the existence of a function which is hard for circuits having oracle gates to* DensityApprox *(see Theorem 6.9) and* Decompress *(see Lemma 6.13). Clearly, these two function problems can be incorporated into a single fixed function problem in* **FNP**.

Remark 9.3. *Observe that one can prove Theorem 9.1 using the alternative assumption that $f\colon \{0,1\}^{\log n} \to \{0,1\}$ requires SVN circuits of size $n^{1-\alpha}$ having a single* DensityApprox *gate.*

Indeed, PRGs allow for a black-box derandomization, and ours allow for a black-box derandomization with only an almost linear slowdown.

Theorem 9.4. *There exist a constant $\widetilde{c} \geq 1$ such that the following holds. Let $L \subseteq \{0,1\}^n$ and $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time $t = t(n) \geq n$ such that for every $x \in \{0,1\}^n$,*

$$\Pr_{y\sim U_t}[A(x,y) \neq L(x)] < \frac{1}{2} - \varepsilon$$

for some constant $\varepsilon > 0$. Assume for every positive integer m there exists a function $f\colon \{0,1\}^m \to \{0,1\}$ computable in $\mathbf{DTIME}(2^{c_f m})$ for some c_f, for which $\mathsf{size}_{\mathbf{FNP}}(f) > 2^{(1-\alpha_0)m}$ for some $\alpha_0 < \frac{1}{c}$. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{c}$.

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time

$$t^{2+\widetilde{c}\alpha} + t_P,$$

where the term $t_P = t^{\gamma(1+\widetilde{c}\alpha)}$ for $\gamma = \max\{2+\widetilde{c}\alpha, c_f+1\}$ corresponds to a step that can be precomputed for all algorithms with running time t. That is, the slowdown of every randomized algorithm running in time t, under our complexity-theoretic assumptions, is at most $t^{1+O(\alpha)}$.

Proof: Let c be the constant guaranteed to us by Theorem 9.1 and set $c' = \frac{c+1}{1-c^2\alpha} > 0$. Set $m = \log(t^{1+c'\alpha})$, $M = 2^m$, and let $f\colon \{0,1\}^{\log(t^{1+c'\alpha})} \to \{0,1\}$ be the guaranteed hard function. Let

$$\overline{G}^f\colon \{0,1\}^{d=(1+c\alpha)\log s} \to \{0,1\}^s$$

be the ε-PRG fooling circuits of size s guaranteed to us by Theorem 9.1, with $s = M^{1-c\alpha}$. We note that $s \geq t^{1+\alpha} = \omega(t\log t)$. Furthermore, we can see that $d \leq (1+c'\alpha)\log t$ since $d = (1+c\alpha)(1-c\alpha)\log M < \log M$, and $(1+c'\alpha)\log t = \log M$.

Consider truncating the output length of $\overline{G}^f$ down to length t. Fix some $x \in \{0,1\}^n$ and let $C_x\colon \{0,1\}^t \to \{0,1\}$ be the circuit that computes $A(x,\cdot)$, of size s. By the properties of the PRG,

$$\left|\Pr[C_x(U_t)=1] - \Pr[C_x(\overline{G}^f(U_d))=1]\right| \leq \varepsilon,$$

so for $x \in L$, $\Pr[C_x(\overline{G}^f(U_d))=1] > \frac{1}{2}$ and for $x \notin L$, $\Pr[C_x(\overline{G}^f(U_d))=1] < \frac{1}{2}$. Hence, the standard way of constructing A_{D} would be to first compute the set

$$I = \left\{\overline{G}^f(z) : z \in \{0,1\}^d\right\},$$

which is independent of C_x so can be though of as a preprocessing step for all circuits of a certain size, and then run $A(x,y)$ for every $y \in I$. A_{D} would then accept if and only if the majority of runs returned 1. The parameters immediately follow.

The time t_P represents the time it takes to compute the support of the PRG. As Theorem 9.1 states, computing the support of the PRG takes time $s^{\frac{\gamma'}{1-c\alpha}} = t^{\gamma'(1+c'\alpha)}$, for $\gamma' = \{2+c\alpha, c_f+1\}$. The theorem holds by setting $\widetilde{c} = \max\{c^2, c', 7\}$. ■

9.1 On Derandomizing AM

Our PEGs output pseudoentropy that fools certain classes of nondeterministic circuits. Hence, it is natural to ask whether we can take advantage of that to derandomize **AM**. We recall that $L \in \mathbf{AMTIME}(t)$ for $t = t(n)$ if there exists a deterministic TM $M\colon \{0,1\}^n \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}$ running in time t such that on input $x \in \{0,1\}^L$, if $x \in L$ then

$$\Pr_{y\sim U_{t(n)}}\left[\exists z \in \{0,1\}^{t(n)}, M(x,y,z)=1\right] \geq \frac{2}{3}$$

and if $x \notin L$ then

$$\Pr_{y\sim U_{t(n)}}\left[\exists z \in \{0,1\}^{t(n)}, M(x,y,z)=1\right] \leq \frac{1}{3}.$$

And so, $\mathbf{AM} = \bigcup_c \mathbf{AMTIME}(n^c)$.

Klivans and van Melkebeek [KvM02] showed that one can use a PRG fooling circuits with **NP** gates to derandomize **AM**, putting it in **NP**. This adds to a line of research trying to find the minimal conditions under which a derandomization of **AM** is possible (see also [AK97, MV05, SU05]). We revisit their proof and show that our PRG from Section 9 can derandomize **AM** with a smaller slowdown than was previously known. Note that since we allow nondeterminism, we can relax our condition on f and allow it to be in $\mathbf{NE} \cap \mathbf{coNE}$.

Theorem 9.5. *There exists a constant $\widetilde{c} \geq 1$ such that the following holds. Let n be a positive integer, $t = t(n)$, and assume that for every positive integer m there exists a function $f\colon \{0,1\}^m \to \{0,1\}$ computable in $\mathbf{NTIME}(\widetilde{O}(2^m)) \cap \mathbf{coNTIME}(\widetilde{O}(2^m))$ that requires $\mathbf{FP}^{\mathbf{NP}}$-circuits of size $2^{(1-\alpha_0)m}$ for some constant $\alpha_0 < \frac{1}{\widetilde{c}}$. Fix any α such that $\alpha_0 < \alpha \leq \frac{1}{\widetilde{c}}$. Then,*

$$\mathbf{AMTIME}(t) \subseteq \mathbf{NTIME}\left(t^{4+3\widetilde{c}\alpha}\right).$$

Proof (sketch): Given $L \in \mathbf{AMTIME}(t)$ with a corresponding TM M, define

$$L' = \left\{(x,y) \in \{0,1\}^n \times \{0,1\}^t : \exists z \in \{0,1\}^t, M(x,y,z) = 1\right\},$$

and observe that $x \in L$ implies that $\Pr_{y \sim U_t}[y \in L'(x,\cdot)] \geq \frac{2}{3}$ and $x \notin L$ implies that $\Pr_{y \sim U_t}[y \in L'(x,\cdot)] \leq \frac{1}{3}$. Since $L' \in \mathbf{NTIME}(O(t))$, by Cook's theorem [Coo88] there exists a circuit $C\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ that computes L', has size $s = \widetilde{O}(t)$ and has a single SAT gate. For every $x \in \{0,1\}^n$, let $C_x\colon \{0,1\}^t \to \{0,1\}$ be the circuit that is obtained by hardwiring x in C.

Next, we argue that there exists a PRG

$$\overline{G}^f\colon \{0,1\}^d \to \{0,1\}^t$$

that $\frac{1}{6}$-fools C_x, with $d = (1+\widetilde{c}\alpha)\log t$. Towards that goal, we extend Claim 9.6.

Claim 9.6. *Define* $\mathrm{DensityApprox}^{\mathrm{SAT}}$ *as before, but now where we allow C to have a* SAT *oracle gate. Assume there exists a function in* **E** *that requires exponential-size nondeterministic circuits. Then, for every constant $\varepsilon > 0$,* $\mathrm{DensityApprox}^{\mathrm{SAT}} \in \mathbf{FP}^{\mathbf{NP}}$.

Thus, Theorem 6.9 also holds for the distinguisher C_x if we also allow $\mathbf{FP}^{\mathbf{NP}}$ gates rather than only **FP** gates. Therefore, we can use the same $\overline{G}^f$ instantiated for Theorem 9.4 with the hard function $f\colon \{0,1\}^m \to \{0,1\}$ where $m \leq \log(t^{1+\widetilde{c}\alpha})$.

The rest of the proof proceeds exactly as in [KvM02]. Using the fact that $f \in \mathbf{NTIME}(\widetilde{O}(2^m)) \cap \mathbf{coNTIME}(\widetilde{O}(2^m))$, they show that in

$$\mathbf{NTIME}\left(O\left(2^d \cdot t \cdot T\right)\right)$$

one can decide whether or not $\Pr[C_x(\overline{G}^f(U_d)) = 1] > \frac{1}{2}$ by supplying the appropriate witnesses, and T is the time it takes to compute, nondeterministically, a single bit of $\overline{G}^f$. For our setting of parameters, $T = \widetilde{O}(2^{2m})$ and

$$\widetilde{O}\left(2^d \cdot t \cdot T\right) = \widetilde{O}\left(t^{1+\widetilde{c}\alpha} \cdot t \cdot t^{2(1+\widetilde{c}\alpha)}\right).$$

■

Remark 9.7. *By strengthening our assumption to $f \in \mathbf{DTIME}(\widetilde{O}(2^m))$ in the above theorem, rather than $f \in \mathbf{NTIME}(\widetilde{O}(2^m)) \cap \mathbf{coNTIME}(\widetilde{O}(2^m))$, one can approximate $\Pr[C_x(\overline{G}^f(U_d)) = 1]$ deterministically, as in Theorem 9.4, and obtain $\mathbf{AMTIME}(t) \subseteq \mathbf{NTIME}\left(t^{2+O(\alpha)}\right)$.*

10 Acknowledgement

We thank Scott Aaronson, Lijie Chen, Amnon Ta-Shma, Roei Tell and Ryan Williams for helpful and interesting discussions. In particular, we thank Scott Aaronson and Lijie Chen for suggesting open problem (5) and Roei Tell for detailed comments and suggestions on a preliminary version of this paper.

References

[AASY16] Benny Applebaum, Sergei Artemenko, Ronen Shaltiel, and Guang Yang. Incompressible functions, relative-error extractors, and the power of nondeterministic reductions. *Computational Complexity*, 25(2):349–418, 2016.

[ABN$^+$92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1978)*, pages 75–83. IEEE, 1978.

[AEL95] Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1995)*, pages 512–519. IEEE, 1995.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[AIKS16] Sergei Artemenko, Russell Impagliazzo, Valentine Kabanets, and Ronen Shaltiel. Pseudorandomness when the odds are against you. In *Proceedings of the 31st Annual Conference on Computational Complexity (CCC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[AK97] Vikraman Arvind and Johannes Köbler. On resource-bounded measure and pseudorandomness. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249. Springer, 1997.

[ALRS98] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):487–510, 1998.

[AS17] Sergei Artemenko and Ronen Shaltiel. Pseudorandom generators with optimal seed length for non-boolean poly-size circuits. *ACM Transactions on Computation Theory (TOCT)*, 9(2):6, 2017.

[BOV07] Boaz Barak, Shien Jin Ong, and Salil Vadhan. Derandomization in cryptography. *SIAM Journal on Computing*, 37(2):380–400, 2007.

[BRSW12] Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl-Wilson construction. *Annals of Mathematics*, 176(3):1483–1544, 2012.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *Approximation, Randomization, and Combinatorial Optimization – Algorithms and Techniques*, pages 200–215. Springer, 2003.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *Annual Cryptology Conference*, pages 151–168. Springer, 2011.

[Coo88] Stephen A. Cook. Short propositional formulas represent nondeterministic computations. *Information Processing Letters*, 26(5):269–270, 1988.

[CS16] Gil Cohen and Leonard J. Schulman. Extractors for near logarithmic min-entropy. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 178–187. IEEE, 2016.

[CT19] Lijie Chen and Roei Tell. Bootstrapping results for threshold circuits "just beyond" known lower bounds. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC 2019)*, pages 34–41. ACM, 2019.

[DHK$^+$19] Irit Dinur, Prahladh Harsha, Tali Kaufman, Inbal Livni Navon, and Amnon Ta Shma. List decoding with double samplers. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2134–2153. SIAM, 2019.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 293–302. IEEE, 2008.

[Dru13] Andrew Drucker. Nondeterministic direct product reductions and the success probability of sat solvers. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 736–745. IEEE, 2013.

[F$^+$03] Joel Friedman et al. Relative expanders or weakly relatively ramanujan graphs. *Duke Mathematical Journal*, 118(1):19–35, 2003.

[FOR15] Benjamin Fuller, Adam O'neill, and Leonid Reyzin. A unified approach to deterministic encryption: New constructions and a connection to computational entropy. *Journal of Cryptology*, 28(3):671–717, 2015.

[FR12] Benjamin Fuller and Leonid Reyzin. Computational entropy and information leakage. *IACR Cryptology ePrint Archive*, 2012:466, 2012.

[FSUV13] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9(26):809–843, 2013.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, July 1998.

[GI01] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001)*, pages 658–667. IEEE, 2001.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 812–821. ACM, 2002.

[GI03] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 126–135. ACM, 2003.

[Gil98] David Gillman. A chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.

[GRS19] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book*, 2019.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *Journal of the ACM (JACM)*, 56(4):20, 2009.

[GW02] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 209–223. Springer, 2002.

[GW14] Oded Goldreich and Avi Widgerson. On derandomizing algorithms that err extremely rarely. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 109–118. ACM, 2014.

[Hea08] Alexander D. Healy. Randomness-efficient sampling within $\mathbf{NC}^1$. *Computational Complexity*, 17(1):3–37, 2008.

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

[HIOS15] Iftach Haitner, Yuval Ishai, Eran Omri, and Ronen Shaltiel. Parallel hashing via list recoverability. In *Annual Cryptology Conference*, pages 173–190. Springer, 2015.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–186. Springer, 2007.

[HRZW17] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes & applications. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2017)*, pages 204–215. IEEE, 2017.

[HW18] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997)*, pages 220–229. ACM, 1997.

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

[KMRZS17] Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *Journal of the ACM (JACM)*, 64(2):11, 2017.

[KU11] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[MRSV19] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil Vadhan. Deterministic approximation of random walks in small space. *arXiv preprint arXiv:1903.06361*, 2019.

[MV05] Peter B. Miltersen and N. Vinodchandran Variyam. Derandomizing Arthur–Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

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

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[RRV02] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. *Journal of Computer and System Sciences*, 65(1):97–128, 2002.

[RSW06] Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. *SIAM Journal on Computing*, 35(5):1185–1209, 2006.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, pages 157–187, 2002.

[RW18] Atri Rudra and Mary Wootters. Average-radius list-recoverability of random linear codes. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 644–662. SIAM, 2018.

[SGP15] Maciej Skorski, Alexander Golovnev, and Krzysztof Pietrzak. Condensed unpredictability. In *International Colloquium on Automata, Languages, and Programming*, pages 1046–1057. Springer, 2015.

[Sha04] Ronen Shaltiel. Recent developments in explicit constructions of extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century Vol 1: Algorithms and Complexity Vol 2: Formal Models and Semantics*, pages 189–228. World Scientific, 2004.

[Sko15] Maciej Skorski. Metric pseudoentropy: Characterizations, transformations and applications. In *International Conference on Information Theoretic Security*, pages 105–122. Springer, 2015.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM (JACM)*, 52(2):172–216, 2005.

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *computational complexity*, 15(4):298–341, 2006.

[SU07] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness vs. randomness tradeoffs for am. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 430–439. ACM, 2007.

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

[Ta-17] Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017)*, pages 238–251. ACM, 2017.

[Tel18] Roei Tell. Quantified derandomization of linear threshold circuits. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 855–865. ACM, 2018.

[Tel19] Roei Tell. Improved bounds for quantified derandomization of constant-depth circuits and polynomials. *computational complexity*, 28(2):259–343, 2019.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[TV00] Luca Trevisan and Salil Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, pages 32–42. IEEE, 2000.

[TZ04] Amnon Ta-Shma and David Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004.

[TZS06] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed–Muller codes. *Journal of Computer and System Sciences*, 72:786–812, 2006.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[Wee04] Hoeteck Wee. On pseudoentropy versus compressibility. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity (CCC 2004)*, pages 29–41. IEEE, 2004.

[Wic13] Daniel Wichs. Barriers in cryptography with weak, correlated and leaky sources. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 111–126. ACM, 2013.

[Wil16] Ryan Williams. Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In *Proceedings of the 31st Annual Conference on Computational Complexity (CCC 2016)*, pages 2:1–2:17, 2016.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 80–91. IEEE, 1982.

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.

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

[Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3:103–128, 2007.

A Condensers and List Recoverable Codes

In this section we continue the discussion of Section 1.3, proving the equivalence between strong condensers and list recoverable codes.

Definition A.1. *Given a function $E\colon \{0,1\}^a \times [n] \to \Sigma$, we denote $\mathcal{C}_E\colon \{0,1\}^a \to \Sigma^n$ as the code mapping $x \in \{0,1\}^a$ to*

$$\mathcal{C}_E(x) = E(x,1) \circ \ldots \circ E(x,n).$$

Note that the rate of $\mathcal{C}_E$ is $\frac{a}{n \log |\Sigma|}$.

Theorem A.2. *Let $\mathsf{Cond}\colon \{0,1\}^a \times [n] \to \Sigma$ be some function so that $\mathcal{C}_{\mathsf{Cond}}$ is (ε, ℓ, L) list recoverable. Then, Cond is a strong*

$$\left(k = \log \frac{L}{\varepsilon}, k' = \log \frac{\ell}{n}, 2\varepsilon\right)$$

condenser. Recall that $\frac{\ell}{n} = 2^{k'}$ is the average size of a list $\mathcal{C}_{\mathsf{Cond}}$ can handle.

Proof: Assume towards a contradiction that Cond is not such a condenser, so there exists an (a,k) source X for which $\mathsf{Cond}(X,Y) \circ Y$ is not ε-close to having min-entropy $k'+d$, where Y is uniformly distributed over $[n]$. By Claim 2.26 there exist sets $S_1, \ldots, S_n \subseteq \Sigma$ satisfying $\sum_{i=1}^n |S_i| \le n \cdot 2^{k'} = \ell$ such that

$$\Pr_{x \sim X, i \sim [n]}[\mathsf{Cond}(X,i) \in S_i] = \Pr_{x \sim X, i \sim [n]}[\mathcal{C}_{\mathsf{Cond}}(x)_i \in S_i] > 2\varepsilon.$$

By an averaging argument, there exists a set $G \subseteq \{0,1\}^a$ of density larger than ε such that for every $x \in G$,

$$\Pr_{i \sim [n]}[C(x)_i \in S_i] \ge \varepsilon.$$

By the list recovery properties of $\mathcal{C}_{\mathsf{Cond}}$ it must hold that $|G| \le L$, however $|G| > \varepsilon \cdot |\mathrm{Supp}(X)| \ge \varepsilon \cdot 2^k \ge L$, in contradiction. ■

Theorem A.3. *Let* $\mathsf{Cond}\colon \{0,1\}^a \times [n] \to \Sigma$ *be a strong* (k, k', ε) *condenser. Then,* $\mathcal{C}_{\mathsf{Cond}}$ *is*

$$\left(4\varepsilon, \ell = 2\varepsilon n \cdot 2^{k'}, L = 2^k\right)$$

list recoverable.

Proof: Let $S_1, \ldots, S_n \subseteq \Sigma$ satisfy $\sum_{i=1}^{n} |S_i| \le \ell$. Define the test $T \subseteq \Sigma \times [n]$ so that $(z, i) \in T$ if and only if $z \in S_i$. Let

$$\mathcal{L} = \left\{ u \in \mathcal{C}_{\mathsf{Cond}} : \Pr_{i \sim [n]} [u_i \in S_i] \ge 4\varepsilon \right\}$$

and assume towards a contradiction that $|\mathcal{L}| \ge 2^k = L$. Note that the set $\mathcal{L}$ is in one-to-one correspondence with the set

$$\mathcal{A} = \left\{ x \in \{0,1\}^a : \Pr_{i \sim [n]} [\mathsf{Cond}(x, i) \in S_i] \ge \frac{\ell}{n \cdot 2^{k'}} + 2\varepsilon \right\}.$$

Let Y be uniformly distributed over $[n]$. Then, on the one hand, $H_\infty^\varepsilon(\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y) \ge k' + \log n$ so by Claim 2.26,

$$\Pr[\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y \in T] \le \varepsilon + |T| \cdot 2^{-k' - \log n} \le \varepsilon + \frac{\ell}{n \cdot 2^{k'}}.$$

On the other hand,

$$\Pr[\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y \in T] = \frac{1}{|\mathcal{A}|} \sum_{x \in \mathcal{A}} \Pr_{i \sim [n]} [\mathsf{Cond}(x, i) \in S_i] \ge \frac{\ell}{n \cdot 2^{k'}} + 2\varepsilon > \frac{\ell}{n \cdot 2^{k'}} + \varepsilon,$$

in contradiction. ■