2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)

Nearly Optimal Pseudorandomness From Hardness

Dean Doron
Department of Computer Science
Stanford University
Stanford, CA, USA
Email: ddoron@stanford.edu

Dana Moshkovitz, Justin Oh, David Zuckerman
Department of Computer Science
University of Texas at Austin
Austin, TX, USA
Email: {danama, sjo, diz}@cs.utexas.edu

***Abstract*—Existing proofs that deduce BPP = 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 randomized single-valued nondeterministic (SVN) 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). The latter derandomization result holds under weaker assumptions, of exponential lower bounds against deterministic SVN circuits.**

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 randomized SVN 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.

***Keywords*-derandomization; pseudorandom generators; list-recoverable codes; pseudo-entropy;**

I. Introduction

A. 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 [1], 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 [2], [3].[1] In other words, under plausible assumptions, $\mathbf{BPP} = \mathbf{P}$. Concretely:

Theorem 1 (previously known derandomization)**.** *Assume there exists a constant $\alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{O(n)}\right)$ such that f requires circuits of size $2^{\alpha n}$ for all sufficiently large n. Let A be a bounded-error probabilistic algorithm, accepting a language[2] 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* $\mathrm{poly}(t, n)$.

Previous works [2]–[5] proving Theorem 1 did not focus on optimizing the runtime of the resulting polynomial time deterministic algorithm. Thus for reasons inherent to the proof techniques of the above works, the runtime of the deterministic algorithm of Theorem 1 is a large polynomial. Indeed, to the best of our knowledge, the best known runtime of the resulting deterministic algorithm is at least $O(t^8)$ [5]. Our main theorem gives a derandomization with running time $t \cdot \max\{t, n\}^{1+\alpha}$ for an arbitrarily small constant $\alpha > 0$. However, our result relies on a few changes to the statement of Theorem 1. The most significant change is that we rely on a somewhat stronger assumption than that of Theorem 1, namely we require that there are no small *randomized single-valued nondeterministic* circuits for f. One can view such circuits as a nonuniform analogue of $\mathbf{MA} \cap \mathbf{coMA}$: the circuit is allowed to additionally take as input a nondeterministic witness and use randomness to verify the witness.

Theorem 2 (efficient general derandomization)**.** *Assume there exists a small constant $0 < \alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ such that f requires randomized single-value nondeterministic circuits of size $2^{(1-\alpha)n}$ for all sufficiently large n. Let A be a bounded-error*

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

[2]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.

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 $t \cdot \max\{t, n\}^{1+O(\alpha)}$.

There are two main differences between the statements of Theorem 1 and Theorem 2.

1) While Theorem 1 requires $f \in \mathbf{DTIME}\left(2^{O(n)}\right)$ that is hard for circuits of size $2^{\Omega(n)}$, we require $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ that is hard for circuits of size $2^{(1-\alpha)n}$. This type of strengthened, plausible, assumption is fundamentally necessary in the hardness vs. randomness paradigm to achieve a low polynomial running time. Moreover, previous proof techniques using our strengthened assumption would still inherently yield a large polynomial running time.
2) We require hardness against randomized single-value nondeterministic circuits, rather than standard ones. The need for this stronger assumption stems from our techniques.

Our derandomized running time is nearly tight for some of the examples above, such as those from property testing. 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 [6] showed that coming up with an $n^{1.999}$-time algorithm, even a nondeterministic one, would refute NSETH[3]. Interestingly, the problem of *multivariate* identity testing also admits a randomized $\widetilde{O}(n)$-time algorithm. Thus, under the complexity-theoretic assumption of Theorem 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 complexity-theoretic assumptions were used before to derandomize both deterministic and nondeterministic classes, as well as to construct various pseudorandomness primitives. Specifically, hardness against randomized SVN circuits was used in [7], [8], and against (deterministic) SVN circuits, which can be seen as the nonuniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$, in [4], [9]. Hardness assumptions against nondeterministic circuits were used in [8]–[13], against circuits with $\mathbf{NP}$ gates in [14], [15] and even against circuits with $\mathbf{PH}$ gates in [16]–[19].

Towards proving our main theorem, we first provide a derandomization of algorithms that err rarely, a scenario known as *quantified derandomization*. First studied by Goldreich and Wigderson [20], in quantified derandomization, the algorithm is assumed to 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.[4]

Theorem 3 (quantified derandomization)**.** *There exists a constant $c \geq 1$ such that the following holds. Assume there exists a constant $\alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ such that f requires requires SVN circuits of size $2^{(1-\alpha)n}$ for all sufficiently large 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 [20]–[23]. We stress that for quantified derandomization, we only assume hardness against (deterministic) SVN circuits, a seemingly much weaker class.

Like Theorem 1, we take the black-box approach for derandomization and prove Theorem 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[5] of the deterministic algorithm, and this number was $\mathrm{poly}(t, n)$ 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. More specifically, consider a hard function f on roughly $\log s$ bits of input, so the truth table of f consists of roughly s bits. Our pseudorandom generator converts those s bits of (worst-case) hardness into s bits of pseudorandomness. This makes it clear that the seed length is essentially optimal.

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

[4]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.

[5]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.

Let us compare this to the prior state-of-the-art PRG given by Umans [5]. There, the seed is of length $c_{\mathsf{U}} \log n$ for a large constant $c_{\mathsf{U}} \geq 8$, and $s = n^{\gamma_{\mathsf{U}}}$ for a small constant $\gamma_{\mathsf{U}} < 1$. Consequently, derandomization using Umans' 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 our PRG 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 3, our quantified derandomization, follows. Informally, a distribution over $\{0,1\}^n$ has pseudoentropy k if it is computationally-indistinguishable, up to some error, from *some* high min-entropy distribution. Thus a PEG is a function $\{0,1\}^d \to \{0,1\}^s$ such that the output distribution (when the input is uniform) has high pseudoentropy.

We mention 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 [24], [25]. The notion of pseudoentropy and pseudoentropy generators was first considered in [26]. Sudan, Trevisan, and Vadhan [27] also construct a pseudoentropy generator under the same notion as in [26]. We construct a pseudoentropy generator under a seemingly weaker definition of pseudoentropy (following Barak et al. [28]) that allows us to get a surprisingly short seed of $O(\alpha) \log n$ for small constant $\alpha > 0$.[6]

In the remainder of the introduction we describe the ideas behind the constructions of our PEG and PRG.

B. Beating the Hybrid Argument

Existing proofs of Theorem 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 [27] constructed PRGs with small seed 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 codes of linear length and a sub-linear number of queries (e.g., codes obtained from Reed-Muller composed with Hadamard, or tensor codes [29]), there are single bit PRGs with a small number of seeds.

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

The large loss in time in Theorem 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. The use of combinatorial designs in [2], [3] inherently doubles the length of the seed. The use of the Reed-Muller code in [4], [5] inherently limits the length of the generator's output. Moreover, the analyses of these constructions go through the infamously-hard-to-beat *hybrid argument* [30]. 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.

More formally, suppose we wish to show that a distribution X on $\{0,1\}^t$ is ε-indistinguishable from uniform for circuits of size s via the hybrid argument. Proceeding with Yao's next bit predictability argument, we must show that for every $i \in [t]$, no circuit $C\colon \{0,1\}^{i-1} \to \{0,1\}$ of size roughly s can predict X_i with probability greater than $\frac{1}{2} + \frac{\varepsilon}{t}$ when fed with $x \sim X_{[1,i-1]}$. Just as one can bound the seed length of an optimal $G\colon \{0,1\}^\ell \to \{0,1\}^t$ that fools circuits of size s using the probabilistic method, one can calculate the seed length of a nonexplicit $G\colon \{0,1\}^\ell \to \{0,1\}^t$ that is ε-unpredictable at all i-s. The calculation shows that such a G has seed length roughly $\log s + 2\log\frac{1}{\varepsilon}$. Thus, for next bit unpredictability error $\frac{\varepsilon}{t}$, the seed length requires at least an additional $2\log t$ bits.

Hence, in order get a PRG with nearly optimal seed, and an efficient derandomization, we must beat the hybrid argument. We do that by first constructing a pseudoentropy generator, 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 [28] 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.

C. Locally Decodable Codes All the Way Down

As explained above, it was known that single bit pseudorandom generators follow from binary locally decodable codes [27]. The construction for obtaining a single bit of pseudorandomness is simple: apply the binary locally decodable code to the truth table of a hard function f, and output a random coordinate of the codeword. However, as discussed above, the method of extending single bit to many bit pseudorandom generators incurs inherent losses in the seed length. In our work we propose a natural deviation from the above idea. We instead apply a code that has *large* alphabet to f and output a random coordinate of this code. The hope is that the symbol at this random coordinate (which, for a large alphabet, will be represented by a large number of bits), will in fact have all the pseudorandom bits we need. However, we cannot guarantee that the random symbol will have "perfect pseudorandomness", i.e. will be computationally close to uniform. Instead, we show that such a random coordinate will have high *pseudoentropy*, i.e will be computationally close to a distribution with at least as much min-entropy as the number of pseudorandom bits we wish to output. In fact, our technique shows that *pseudoentropy generators* that output many bits follow from locally decodable codes over a large alphabet.

More accurately, we consider (a specialized version of) locally *list recoverable* codes. Initially, list recoverable codes were used as an intermediate step for constructing list decodable codes (e.g., in [31]–[33]) but have since gained independent interest, with several applications and dedicated constructions (e.g., [34]–[37]). 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. 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 work with multiple definitions of pseudoentropy, which we now discuss. First, we show a close relation between locally list recoverable codes and *Yao pseudoentropy*. Roughly, a random variable $X \sim \{0,1\}^n$ has high Yao pseudoentropy if there is no small, computationally enumerable subset A of $\{0,1\}^n$ that "explains" too much of X, i.e., $\Pr[X \in A]$ is significant. Now, if we apply a locally list recoverable code with large alphabet to a hard function f and pick a random coordinate, then this random variable must have high Yao pseudoentropy. Otherwise, there would be a small, efficiently computable subset of $\{0,1\}^n$ that explains a significant fraction of the codeword. This roughly corresponds to a small list $S_1 \cup \ldots \cup S_n$ that contains many symbols of the codeword. Hence, f can be efficiently recovered from this list, which is a contradiction.

For the purposes of converting pseudoentropy to pseudorandomness, and other applications, it is convenient to consider other notions of pseudoentropy. The notion of *metric pseudoentropy* was first studied by Barak et al. [28] and later in various works in cryptography [38]–[44]. We say a random variable $X \sim \{0,1\}^n$ has k metric 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 $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \le \varepsilon$. Metric pseudoentropy differs from the widely-used notion of HILL pseudoentropy [26] where the high min-entropy random variable does not depend on the distinguishing circuit. Although seemingly weaker, as an additional application, this definition allows us to derandomize algorithms that err rarely, as random variables with high metric pseudoentropy cannot have large weight on small sets.

We obtain our PEG by first formalizing the above connection between locally list recoverable codes and Yao pseudoentropy [45]. We then construct a locally list recoverable code fitting our purposes. To obtain such a code, we simply "fold" a locally list decodable code $\mathcal{C} \colon \{0,1\}^n \{0,1\}^{\bar{n}}$ (handling constant agreement) so that every $n^{1-O(\alpha)}$ coordinates in $\mathcal{C}$ is considered a single symbol. The resulting code has exponential alphabet size and can handle exponential list size as desired. Moreover, it is decodable with few queries and can also handle a constant fraction of agreement. As one will see in Section III, this gives a PEG (and ultimately a PRG) with constant error, which is sufficient for derandomizing **BPP**.

Finally, we utilize the fact that one can get the (more convenient) metric pseudoentropy from Yao pseudoentropy if we allow the enumeration circuit to be an SVN circuit. As a brief sketch of how to do so, we first mention that if a random variable X has high metric pseudoentropy, then for any efficient distinguisher D, the support of D cannot disproportionately explain X (namely, $\Pr[D(X) = 1]$ is bounded). This is quite close to the definition of Yao pseudoentropy; however, we must give a computationally efficient way to enumerate the support of D. To do so, we follow [28] and hash the support of D onto a smaller universe. The preimage of the hash function is an enumeration of the support of D, provided we can use nondeterminism to guess a preimage (and use D itself to verify whether that preimage is correct). By assuming a function f that is hard for SVN circuits, we get a metric pseudoentropy generator. The rest of the details are given in Section III. We stress that in our analysis of the PEG, nondeterminism is used only to convert Yao pseudoentropy to metric pseudoentropy. Indeed, our PEG outputs high Yao pseudoentropy even under the assumption that f is only hard for standard circuits.

It is interesting to draw an analogy between our pseudoentropy generator and its information-theoretic counterparts. The work of Ta-Shma and Zuckerman [25], following Trevisan [24], 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 [46] 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 $L = 2^k$. Thus, in a way, our pseudoentropy generator is a computational manifestation of these connections.

D. From Pseudoentropy to Pseudorandom Bits

We construct our pseudorandom generator $\overline{G}^f$ by composing our pseudoentropy generator G^f with a new construction of an extractor with seed length close to $\log n$, supporting $n^{1-\alpha}$ min-entropy for any $\alpha < \frac{1}{2}$. This adds to the short list of extractors with almost the right dependence on n, currently including [47], [48] and one-bit extractors coming from good list decodable codes. Our construction essentially follows a construction given in [47]; however, there it was analyzed for smaller min-entropies.

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. It is in this composition argument that we require randomness in the circuit, and thus ultimately in the hardness assumption. See Section V for more details.

E. Reducing the Error

The construction of the locally list recoverable code discussed in Section I-C supports a constant fraction of agreement. This implies that the resulting PEGs and PRGs have constant error. Indeed, this suffices for derandomizing $\mathbf{BPP}$ and for quantified derandomization. Nevertheless, there are several reasons to reduce the error. First, a small error PRG can be used to derandomize algorithms with error approaching $\frac{1}{2}$. Second, often when PRGs are used as components in other pseudorandomness constructions, it is necessary to have small error because of other error losses. Third, it is a natural question on its own, and significant research effort has been devoted to reducing the error in other PRGs. The best error we can hope for is $\varepsilon = n^{-\Omega(1)}$, since we wish to keep the seed length $(1+O(\alpha))\log n$. We manage to obtain this.

To get a PEG with a better dependence on ε, we improve the construction of our locally list recoverable code $\mathcal{C}$ to support ε-fraction of agreement even when the underlying locally list decodable code $\mathcal{C}_{\mathsf{LDC}}$ supports only a constant fraction $\varepsilon_0 \gg \varepsilon$ of agreement. This is done via an expander-based transformation, and the details are given in Section VIII. Getting a PRG with ε error almost readily follows, since the extractor we construct already works for such an error.

F. On Our Hardness Assumption

Finally, we discuss our hardness assumption. In order to obtain our result, there are two ways our assumption is stronger than in previous works. First, we require that f must be hard for circuits of size at least $2^{(1-\alpha)n}$ for some small α, whereas previous constructions only required hardness at least $2^{\beta n}$ for some $\beta < 1$ (and generally β is thought of as small). The first strengthening of the assumption seems necessary for a derandomization as efficient as ours since the hardness should be comparable to the size of circuits we wish to fool.

Second, we require a function f that is hard to compute for circuits that are allowed to use both nondeterminism and randomness. One might argue that in the non-uniform model of circuits, one can convert randomness into advice using an Adleman type argument and incur only a minor loss in circuit size. However, the size blowup of Adleman's theorem in the case of nondeterministic circuits is larger. This, combined with the fact that we require the size of our hardness to be $2^{(1-\alpha)n}$ makes such an argument impossible. Indeed, in order to convert a randomized SVN circuit into an SVN circuit, one must union bound over all possible inputs and all possible witness strings. Generally, the witness length can be nearly as large as the circuit itself. In other words, we would need an error probability close to 2^{-s} where s is the size of the circuit. Achieving that requires repeating the circuit nearly s times, incurring a quadratic blowup and making our hardness assumption infeasible as we initially assume hardness against randomized SVN circuits of size $2^{(1-\alpha)n}$. In summary, previous works have similarly used assumptions in which the function f is hard against models of computation that use nondeterminism and randomness. However, we work on the very "high-end" of the hardness assumptions, i.e., assume hardness for very large circuits, and this seems necessary for our proof.

We now discuss the plausibility of our assumption. Our result relies on the assumption that there exists a function f satisfying two competing properties. First, it should be computable by a deterministic TM in time $2^{(1+\alpha)n}$, for some small constant α. Second, it should also be hard for circuits of size $2^{(1-\alpha)n}$ that use nondeterminism, as well as randomness, to verify the nondeterministic witness. We hence need a function that resides in $\mathbf{E}$ but is still "difficult enough" for a non-uniform Merlin-Arthur proof system to verify (i.e., requiring Arthur to run in time at least $2^{(1-\alpha)n}$). Thus, our assumption is plausible if random verification doesn't always achieve significant savings.

Sometimes, random verification does yield savings. For example, Williams [6] proved that a variety of $\#\mathbf{P}$-complete problems have MA proof systems that run in time 2^{cn} for various $c < 1$, whereas the best known deterministic algorithms solving these problems take time $O(2^n)$. He also gave a 3-round interactive proof system running in time $2^{.67n}$ time for QBF, a $\mathbf{PSPACE}$-complete problem. Even if these results make one worry about finding our desired f in $\mathbf{PSPACE}$, it is widely believed that there exists problems computable in time $2^{(1+\alpha)n}$ that require exponential space. Given that such problems lack the structure that Williams exploits, it is plausible that there is such an f where random verification doesn't achieve significant savings, making our hardness assumption true.

G. Open Problems

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

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 SVN – rather than randomized SVN – circuits?
3) 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 [49]. 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.

We omit the proofs of all theorems in this extended abstract. However, full proofs and more in depth discussion can be found in the full version [50].

II. Preliminaries

A. Circuits, Nondeterministic Circuits and Worst-Case Hardness

Definition 4 (SVN circuit). *Let $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\bot\}$ be a partial function. A* single-valued nondeterministic (SVN) circuit *computing f is a pair of circuits $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}}\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ such that for input string $x \in \{0,1\}^n$ and witness string $y \in \{0,1\}^w$, the following holds.*

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

The size of an SVN circuit is the sum of the sizes of C and C_{check}.

When only considering SVN circuits that compute total functions with a single bit of output, the above definition can be viewed as the non-uniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$. We often refer to $C_{\mathsf{check}}(x, y)$ as the checking phase, or witness verification circuit, and refer to $C(x, y)$ as the computing circuit. Additionally, we often think of an SVN circuit as a single circuit combining both C and C_{check} (with $m + 1$ output bits).

Definition 5 (hardness against SVN circuits). *Let $f\colon \{0,1\}^n \to \{0,1\}$. We let $\mathsf{size}_{SVN}(f)$ denote the size of the smallest SVN circuit that computes f. We say f requires exponential-size SVN circuits if $\mathsf{size}_{SVN}(f) > 2^{\delta n}$ for some constant $\delta > 0$.*

We also consider SVN circuits where the checking phase may use randomness. When considering total functions with a single bit output, this model can be seen as the nonuniform analogue of $\mathbf{MA} \cap \mathbf{coMA}$.[7]

Definition 6 (randomized SVN circuits). *Let $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\bot\}$ be a partial function. A* randomized SVN circuit *computing f with error $0 \leq \delta < \frac{1}{2}$ is a pair of circuits $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}}\colon \{0,1\}^n \times \{0,1\}^w \times \{0,1\}^d \to \{0,1\}$ such that for input string $x \in \{0,1\}^n$, witness string $y \in \{0,1\}^w$ and randomness string $r \in \{0,1\}^d$, the following holds.*

1) *For every $x \in \{0,1\}^n$, $f(x) \neq \bot$ if and only if there exists $y \in \{0,1\}^w$ such that*
$$\Pr_{r \sim U_d}[C_{\mathsf{check}}(x, y, r) = 1] \geq 1 - \delta.$$
2) *For every $x \in \{0,1\}^n$ such that $f(x) \neq \bot$, and $y \in \{0,1\}^w$, either $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x, y, r) = 1] \geq 1 - \delta$ or $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x, y, r) = 1] \leq \delta$.*
3) *For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^w$ such that $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x, y, r) = 1] \geq 1 - \delta$, it holds that $C(x, y) = f(x)$.*

The size of a randomized SVN circuit is the sum of the sizes of C and C_{check}.

We can now define hardness for randomized SVN circuits in the natural way.

Definition 7 (hardness for randomized SVN circuits). *We say $f\colon \{0,1\}^n \to \{0,1\}$ requires randomized SVN circuits for error $\delta = \delta(n) > 0$ of size $s = s(n)$ if the smallest randomized SVN circuit that computes f with error δ has size at least s.*

It will be useful to consider circuits in a related model where the checking phase is deterministic but has access to an oracle that gets a circuit and approximates the fraction of inputs the circuit accepts.

Definition 8 ($\mathrm{DensityApprox}_\eta$ gates). *For a fixed error parameter $\eta > 0$, a $\mathrm{DensityApprox}_\eta$ oracle gate takes as input an encoding of a circuit C and outputs $p \in \{0,1\}^{\lceil \log \frac{1}{\eta} \rceil}$, such that $|\mathbb{E}[C] - p| \leq \eta$. In words, the gate outputs an additive η-approximation to the fraction of accepting inputs to C.*

B. Error Correcting Codes

In this work we mainly consider locally list recoverable codes.

Definition 9 (locally list recoverable code). *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 $(Q, \varepsilon, \ell, s)$ locally list recoverable if the following holds. Let $\bar{S} = (S_1, \ldots, S_n)$ be any list of subsets, where*

[7]An alternative definition for randomized SVN circuits was given in [7], [8], corresponding to the nonuniform analogue of $\mathbf{AM} \cap \mathbf{coAM}$. Although one can suspect these models to be equivalent, there are some subtleties that arise when one cares about polynomial blowups in size.

each $S_z \subseteq \Sigma$ and $\sum_{z=1}^{n} |S_z| \leq \ell$, and let $\mathcal{O}_{\bar{S}}$ be any oracle to $\bar{S}$ satisfying the following properties:

- *$\mathcal{O}_{\bar{S}}$ takes as input $y \in \{0,1\}^{\log \ell}$ and outputs an element of $\Sigma \cup \{\perp\}$.*
- *For every $z \in [n]$ and every $v \in S_z$, there exists a $y_{(z,v)} \in \{0,1\}^{\log \ell}$ such that $\mathcal{O}_{\bar{S}}(y_{(z,v)}) = v$. Furthermore, each $y_{(z,v)}$ is unique. That is, $y_{(z,v)} \neq y_{(z',v')}$ if $(z,v) \neq (z',v')$.*

Then there exist circuits $A_1, \dots, A_L$ for some arbitrary L, each of size s, that satisfy the following.

- *Each A_j takes as input $i \in [k]$ and uses at most Q oracle gates to $\mathcal{O}_{\bar{S}}$.*
- *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 $A_j(i) = x_i$. Moreover, A_j never queries $\mathcal{O}_{\bar{S}}$ on an input that yields $\perp$.*

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.

One may notice that our definition of LLRC is a bit specialized towards our goals. 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.

C. Random Variables, Min-Entropy, Extractors

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 X is ε-close to Y and denote it by $X \approx_\varepsilon Y$. We denote by U_n the random variable distributed uniformly over $\{0,1\}^n$.

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

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

Definition 10 (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 $(\mathsf{Ext}(X,Y), Y) \approx_\varepsilon (U_m, Y)$.

Another useful object in our constructions is the *density sampler.*

Definition 11 (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*
$$|\{x \in [N] : x \text{ is } \varepsilon\text{-bad for } B\}| < \delta N.$$

D. Pseudoentropy

We focus on two notions of pseudoentropy in this work. The first resembles the natural definition one might first consider for pseudoentropy (which was used in [26]), except the quantifiers are reversed

Definition 12 (metric pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, 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 $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

The following characterization of metric pseudoentropy is useful.

Lemma 13 ([28]). *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, and $\varepsilon > 0$. Then, $H^{\mathrm{metric}}_{s,\varepsilon}(X) \geq k$ if and only if 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 let D to come from a class of circuits closed under complement, including circuits using* $\mathrm{DensityApprox}_\eta$ *oracle gates.*

A different definition for computational entropy was given by Yao [45] who used *compression* rather than *indistinguishability*.

We extend Yao's definition to SVN circuits.

Definition 14 (NYao set). *For some positive integers s, n, ℓ and $A \subseteq \{0,1\}^n$, we say that $A \in \mathcal{C}^{\mathrm{NYao}}_{\ell,s}$ if there exists a circuit $c\colon \{0,1\}^n \to \{0,1\}^\ell$ and an SVN circuit $d\colon \{0,1\}^\ell \times \{0,1\}^{w'} \to \{0,1\}^\ell$ computing a partial function $f_d\colon \{0,1\}^\ell \to \{0,1\}^n \cup \{\perp\}$, both of size s, such that*

$$A = \{x : f_d(c(x)) = x\}.$$

We refer to c as the compressing circuit and to d as the decompressing one.

Definition 15 (NYao pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, and $\varepsilon > 0$. We say that $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$ if for every $\ell < k$ and $A \in \mathcal{C}^{\mathrm{NYao}}_{\ell,s}$ it holds that $\Pr[X \in A] \leq 2^{\ell - k} + \varepsilon$.*

The following lemma provides a useful connection between NYao and metric pseudoentropy.

Lemma 16 (following [28]). *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 $s \geq s_0$, $H^{\text{NYao}}_{s,\varepsilon}(X) \geq k$ implies $H^{\text{metric}}_{\gamma s,\varepsilon}(X) \geq \frac{k}{2}$.*

E. 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 17 (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.

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

One can show that for $k = n$ the above definition coincides with the standard definition of PRGs [51].

III. A 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. As discussed in the introduction, 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, where we identify f with its truth table in $\{0,1\}^n$. This then implies that it must also have high metric pseudoentropy by Lemma 16.

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, $\mathsf{size}_{SVN}(f) \geq n^{1-\alpha_0}$. 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

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

Theorem 19. *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) metric PEG for $k = \frac{\log \ell}{2}$ and $s = O\left(\frac{n^{1-\alpha}}{Q}\right)$.*

The important points to note about the above theorem is that the pseudoentropy k of the output distribution corresponds to the size of the lists, the agreement ε that the code can handle is equivalent to the error of the pseudoentropy, and if Q is small, the hardness s of f corresponds to the hardness of the output distribution. We can show there exists a code with the desired parameters.

Theorem 20. *For any positive integer n and any constants $0 < \varepsilon < 1$ and $0 < \alpha \leq \frac{1}{6}$, there is a code $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ with $\log|\Sigma| = \widetilde{O}(n^{1-5\alpha})$, and $m = \widetilde{O}(n^{5\alpha})$ that is $(Q,\varepsilon,\ell,s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\ell = 2^{n^{1-8\alpha}}$ and $s_{\mathcal{C}} = n^{1-\alpha}$.*

The idea behind the construction is relatively simple. We modify a constant rate locally list decodable code (that can handle ε agreement) by concatenating every $n^{1-O(\alpha)}$ consecutive symbols into a single symbol. This results in a code with about $n^{O(\alpha)}$ coordinates and about $n^{1-O(\alpha)}$ bits per coordinate. An ε agreement in the consolidated symbols implies an ε agreement in the original LDC symbols. The important points to note are that the number of seeds of the PEG is equal to the number of coordinates in the code, and the output length is equal to the number of bits used per coordinate.

Combining Theorem 20 with Theorem 19 gives the following.

Theorem 21. *For any 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 $\alpha_0 < \frac{1}{7}$. Let α be any constant such that $\alpha_0 < \alpha < \frac{1}{7}$. Then, there exists a function

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

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 $a = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n)$ time 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 $\widetilde{O}(n^{c_f+1})$.

IV. Derandomizing Algorithms That Err Rarely

Intuitively, a distribution with high pseudoentropy is computationally indistinguishable from a distribution with a large support. Moreover, a distribution with large support will contain many strings that lead to a correct answer for an algorithm that errs rarely. Thus our PEG is sufficient to provide a derandomization of such algorithms.

Lemma 22. *There exists a constant $c \geq 1$ such that the following holds. For positive integers n and $t \geq n$,*[8] *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 $\mathrm{Supp}(G)$ and can be precomputed for all algorithms with running time t.

Combining Lemma 22 and Theorem 21, we get the following corollary.

Corollary 23. *There exists a constant $\widetilde{c} \geq 1$ such that the following holds for all positive integers n and $t \geq n$. Assume that for every positive integer w there exists a function $f\colon \{0,1\}^w \to \{0,1\}$ computable in deterministic time $2^{c_f w}$ for some $c_f \geq 1$, for which $\mathsf{size}_{SVN}(f) > 2^{(1-\alpha_0)w}$ for some universal 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 $t^{O(\alpha)}$.

V. Extracting Randomness from Pseudoentropy

We now discuss how to extract pseudorandomness from metric pseudoentropy. To do so, we require hardness against circuits with additional power. Recall the definition of DensityApprox gates from Definition 8. Let metric* be as in Definition 12, except the distinguisher D is now allowed to use DensityApprox gates as well. We denote $H^{\text{metric}^\star}_{s,\varepsilon}(X)$ as the metric* pseudoentropy.

[8]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.

Given a distribution that is hard for distinguishers with oracle access to $\mathrm{DensityApprox}_\varepsilon$ gates, extraction yields pseudorandomness.

Lemma 24. *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 $(k - \log\frac{1}{\varepsilon}, \varepsilon)$ 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 $H^{\text{metric}^\star}_{s',\varepsilon}(X) \geq k$ for $s' = \widetilde{O}(s+t)$. Then, $\mathsf{Ext}(X, U_d)$ 10ε-fools circuits of size s.*

The proof idea for this is relatively simple. Consider any distinguisher A for the final output distribution. Since an extractor is a sampler, there is only a small set $B \subset \{0,1\}^n$ of inputs to the extractor for which for every $x \in B, |\mathbb{E}_{y\sim U_d}[A(\mathsf{Ext}(x,y))] - \mathbb{E}_{z\sim U_m}[A(z)]| \geq \varepsilon$. If A distinguished $\mathsf{Ext}(X, U_d)$ from uniform, then the set B must have been overrepresented in the support of X. Thus, the distinguisher that outputs 1 on B is a contradiction to the characterization of metric pseudoentropy given in Lemma 13. To construct such a distinguisher, we use the $\mathrm{DensityApprox}_\varepsilon$ gate to find the density of $A(\mathsf{Ext}(x,\cdot))$ for any x, and compare it to the value $\mathbb{E}_{z\sim U_m}[A(z)]$, which can be hardwired as advice. Finally, we note that in order to construct a PEG that outputs high metric* pseudoentropy using our techniques, it suffices to add to the hardness assumption that f is hard for circuits with oracle access to $\mathrm{DensityApprox}_\varepsilon$ gates.

VI. Extractors With Near-Optimal Seed Length

Since our pseudoentropy generator has an incredibly small seed length of $O(\alpha)\cdot\log n$, in order to obtain our final PRG with nearly optimal seed length, it suffices to construct an extractor with near-optimal (in n) seed that supports min-entropy $n^{1-\alpha}$ for every $\alpha < \frac{1}{2}$. Specifically, for $\varepsilon = n^{-o(1)}$, we construct an extractor with seed length $(1+O(\alpha))\log n$. A very similar construction was given in [47], but there the analysis was only done for a constant entropy rate.

Theorem 25. *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 $m = \frac{1}{c}n^{1-2\alpha}$.*

VII. PRGs With Nearly Optimal Slowdown

We are now ready to present our final PRG. We first note that although we mentioned in Section V that we use hardness against circuits with oracle access to $\mathrm{DensityApprox}_\varepsilon$ gates, one can simulate the output of such a gate with random sampling (with some error probability). Thus, we state our final PRG theorem in terms of hardness against randomized SVN circuits.

Theorem 26. *There exists a constant $c \geq 7$ such that the following holds for every positive integer n and a constant $\varepsilon > 0$. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ requires randomized SVN circuits for error δ, of size $n^{1-\alpha_0}$ for some universal constant $\alpha_0 < \frac{1}{c}$, and $\delta = 2^{-\log^{c'} n}$ for an arbitrary constant 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 $\gamma = \max\{2+c\alpha, c_f+1\}$.

Theorem 2 readily follows from the above PRG.

VIII. Lower Error PEGs and PRGs

By constructing a locally list recoverable code with smaller error, we can get PEGs and PRGs with error $n^{-\gamma}$ for a small constant γ. Overall, the resulting PEGs and PRGs have seed lengths $O(\gamma) \cdot \log n$ and $(1 + O(\gamma)) \log n$, respectively. Such an error is optimal, up to the constant multiplicative factors, as the seed length must be at least $\log(1/\varepsilon)$ (and for PRGs, at least $\log(n/\varepsilon)$).

As the extractor presented in Section VI already achieves $n^{-\gamma}$ error for a sufficiently small γ, the main challenge is to construct a list recoverable code that can handle $n^{-\gamma}$ *agreement*. As before, we take the length-n truth table of a hard function f, encode it via a locally list decodable code, and consolidate the symbols of the code into $n^{O(\alpha)}$ larger symbols of size $n^{1-O(\alpha)}$ each. Previously, we showed that by simply concatenating consecutive symbols (or, "folding"), an ε agreement in the consolidated symbols implies an ε agreement in the original LDC symbols. The same technique does not work when $\varepsilon = n^{-\gamma}$, as current locally decodable codes have bad dependence on ε and so we incur a large blowup in parameters.

The idea is to instead consolidate symbols in such a manner that wherever the small $n^{-\gamma}$ fraction of good larger symbols might be, at least $1/3$ fraction of the original symbols are contained in at least one of the good larger symbols. Samplers with $n^{-\gamma}$ error naturally satisfy such a requirement (in fact, even their one-sided version, called hitters, suffices). For this to make sense, we need the decoder circuit to be hardwired with information about which lists are good. This is yet another place where we leverage the non-uniform nature of our decoding, recalling that the good lists are only a function of the hard function itself.

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 [32], [33], [52]–[55].

Theorem 27. *There exists a constant $b > 1$ such that the following holds. For any positive integer n, constant $0 < \alpha \leq \frac{1}{7}$, and $\gamma = \frac{\alpha}{b}$, there exists a code $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ with $\log|\Sigma| = \widetilde{O}(n^{1-5\alpha})$ and $m = \widetilde{O}(n^{6\alpha})$ that is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\varepsilon = \frac{2}{n^\gamma}$, $\ell = 2^{n^{1-8\alpha}}$ and $s_{\mathcal{C}} = n^{1-\alpha}$.*

Again, in the same vein as before, the existence of such a code implies the desired PEG and PRG.

Corollary 28. *There exists a constant $b > 1$ such that the following holds. For any positive integer n, assume*

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

is such that $\mathsf{size}_{SVN}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{8}$. Let α be any constant such that $\alpha_0 < \alpha < \frac{1}{8}$ and let $\gamma = \frac{\alpha}{b}$. Then, there exists a function

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

that is a (k, s, ε) SVN metric PEG for $\varepsilon = \frac{2}{n^\gamma}$, $k = \frac{n^{1-8\alpha}}{2}$, $s \geq n^{1-4\alpha}$, $d = 6\alpha \log n + O(\log\log n)$ and $a = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n)$ time 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 $\widetilde{O}(n^{c_f+1+\alpha})$.

Theorem 29. *There exists a constant $c \geq 7$ such that the following holds for every positive integer n. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ requires randomized SVN circuits for error δ, of size $n^{1-\alpha_0}$ for some universal constant $\alpha_0 < \frac{1}{c}$, and $\delta = 2^{-\log^{c'} n}$ for an arbitrary constant c'. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{c}$. Let $\gamma = \frac{\alpha}{c}$, and let $\varepsilon = c \cdot n^{-\gamma}$. 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 $s^{\frac{\zeta}{1-c\alpha}}$ for $\zeta = \max\{2+c\alpha, c_f+1\}$.

Acknowledgment

We are deeply thankful to Ronen Shaltiel for pointing out an error in a previous version, for suggesting an elegant way of significantly simplifying the contents of Section V, and for very useful discussions about hardness assumptions. We are also grateful to Noga Ron-Zewi for valuable discussions on list-recoverable codes that led to a simpler construction. We thank Roei Tell for detailed comments and suggestions on a preliminary version of this paper. Finally, we thank Scott Aaronson and Lijie Chen for suggesting open problem (3), and Omer Reingold, Amnon Ta-Shma, and Ryan Williams for helpful and interesting discussions.

Dean Doron is supported by NSF Grant CCF-1763311, and most of this work was done while at UT Austin, supported by NSF Grant CCF-1705028. Dana Moshkovitz is supported in part by NSF Grant CCF-1705028 and CCF-1648712. Justin Oh is supported by NSF Grant CCF-1705028. David Zuckerman is supported in part by NSF Grant CCF-1705028 and a Simons Investigator Award (#409864).

References

[1] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 653–750, Jul. 1998.

[2] N. Nisan and A. Wigderson, "Hardness vs randomness," *Journal of Computer and System Sciences*, vol. 49, no. 2, pp. 149–167, 1994.

[3] R. Impagliazzo and A. 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)*. ACM, 1997, pp. 220–229.

[4] R. Shaltiel and C. Umans, "Simple extractors for all min-entropies and a new pseudorandom generator," *Journal of the ACM (JACM)*, vol. 52, no. 2, pp. 172–216, 2005.

[5] C. Umans, "Pseudo-random generators for all hardnesses," *Journal of Computer and System Sciences*, vol. 67, no. 2, pp. 419–440, 2003.

[6] R. 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)*, 2016, pp. 2:1–2:17.

[7] D. Gutfreund, R. Shaltiel, and A. Ta-Shma, "Uniform hardness versus randomness tradeoffs for Arthur-Merlin games," *Computational Complexity*, vol. 12, no. 3-4, pp. 85–130, 2003.

[8] R. Shaltiel and C. Umans, "Low-end uniform hardness vs. randomness tradeoffs for AM," in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*. ACM, 2007, pp. 430–439.

[9] P. B. Miltersen and N. V. Variyam, "Derandomizing Arthur–Merlin games using hitting sets," *Computational Complexity*, vol. 14, no. 3, pp. 256–279, 2005.

[10] V. Arvind and J. Köbler, "On resource-bounded measure and pseudorandomness," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 1997, pp. 235–249.

[11] R. Shaltiel and C. Umans, "Pseudorandomness for approximate counting and sampling," *Computational Complexity*, vol. 15, no. 4, pp. 298–341, 2006.

[12] B. Barak, S. J. Ong, and S. Vadhan, "Derandomization in cryptography," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 380–400, 2007.

[13] A. 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)*. IEEE, 2013, pp. 736–745.

[14] A. R. Klivans and D. van Melkebeek, "Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses," *SIAM Journal on Computing*, vol. 31, no. 5, pp. 1501–1526, 2002.

[15] O. Goldreich and A. Wigderson, "Derandomization that is rarely wrong from short advice that is typically good," in *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 2002, pp. 209–223.

[16] L. Trevisan and S. Vadhan, "Extracting randomness from samplable distributions," in *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000)*. IEEE, 2000, pp. 32–42.

[17] S. Artemenko, R. Impagliazzo, V. Kabanets, and R. 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.

[18] B. Applebaum, S. Artemenko, R. Shaltiel, and G. Yang, "Incompressible functions, relative-error extractors, and the power of nondeterministic reductions," *Computational Complexity*, vol. 25, no. 2, pp. 349–418, 2016.

[19] S. Artemenko and R. Shaltiel, "Pseudorandom generators with optimal seed length for non-boolean poly-size circuits," *ACM Transactions on Computation Theory (TOCT)*, vol. 9, no. 2, p. 6, 2017.

[20] O. Goldreich and A. Widgerson, "On derandomizing algorithms that err extremely rarely," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*. ACM, 2014, pp. 109–118.

[21] R. Tell, "Quantified derandomization of linear threshold circuits," in *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*. ACM, 2018, pp. 855––865.

[22] ——, "Improved bounds for quantified derandomization of constant-depth circuits and polynomials," *Computational Complexity*, vol. 28, no. 2, pp. 259–343, 2019.

[23] L. Chen and R. 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)*. ACM, 2019, pp. 34–41.

[24] L. Trevisan, "Extractors and pseudorandom generators," *Journal of the ACM*, vol. 48, no. 4, pp. 860–879, 2001.

[25] A. Ta-Shma and D. Zuckerman, "Extractor codes," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3015–3025, 2004.

[26] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[27] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the XOR lemma," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 236–266, 2001.

[28] B. Barak, R. Shaltiel, and A. Wigderson, "Computational analogues of entropy," in *Approximation, Randomization, and Combinatorial Optimization – Algorithms and Techniques*. Springer, 2003, pp. 200–215.

[29] S. Yekhanin, "Locally decodable codes," *Foundations and Trends in Theoretical Computer Science*, vol. 6, no. 3, pp. 139–255, 2012.

[30] B. Fefferman, R. Shaltiel, C. Umans, and E. Viola, "On beating the hybrid argument," *Theory of Computing*, vol. 9, no. 26, pp. 809–843, 2013.

[31] V. Guruswami and P. Indyk, "Expander-based constructions of efficiently decodable codes," in *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001)*. IEEE, 2001, pp. 658–667.

[32] ——, "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)*. ACM, 2002, pp. 812–821.

[33] ——, "Linear time encodable and list decodable codes," in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*. ACM, 2003, pp. 126–135.

[34] I. Haitner, Y. Ishai, E. Omri, and R. Shaltiel, "Parallel hashing via list recoverability," in *Annual Cryptology Conference*. Springer, 2015, pp. 173–190.

[35] B. Hemenway, N. Ron-Zewi, and M. Wootters, "Local list recovery of high-rate tensor codes and applications," *SIAM Journal on Computing*, vol. 49, no. 4, pp. FOCS17–157–FOCS17–195, 2019.

[36] B. Hemenway and M. Wootters, "Linear-time list recovery of high-rate expander codes," *Information and Computation*, vol. 261, pp. 202–218, 2018.

[37] A. Rudra and M. Wootters, "Average-radius list-recoverability of random linear codes," in *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. SIAM, 2018, pp. 644–662.

[38] S. Dziembowski and K. Pietrzak, "Leakage-resilient cryptography," in *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*. IEEE, 2008, pp. 293–302.

[39] K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz, "Memory delegation," in *Annual Cryptology Conference*. Springer, 2011, pp. 151–168.

[40] B. Fuller and L. Reyzin, "Computational entropy and information leakage." *IACR Cryptology ePrint Archive*, vol. 2012, p. 466, 2012.

[41] D. Wichs, "Barriers in cryptography with weak, correlated and leaky sources," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. ACM, 2013, pp. 111–126.

[42] M. Skorski, "Metric pseudoentropy: Characterizations, transformations and applications," in *International Conference on Information Theoretic Security*. Springer, 2015, pp. 105–122.

[43] B. Fuller, A. O'neill, and L. Reyzin, "A unified approach to deterministic encryption: New constructions and a connection to computational entropy," *Journal of Cryptology*, vol. 28, no. 3, pp. 671–717, 2015.

[44] M. Skorski, A. Golovnev, and K. Pietrzak, "Condensed unpredictability," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2015, pp. 1046–1057.

[45] A. C. Yao, "Theory and application of trapdoor functions," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*. IEEE, 1982, pp. 80–91.

[46] V. Guruswami, C. Umans, and S. Vadhan, "Unbalanced expanders and randomness extractors from parvaresh–vardy codes," *Journal of the ACM (JACM)*, vol. 56, no. 4, p. 20, 2009.

[47] A. Ta-Shma, D. Zuckerman, and S. Safra, "Extractors from Reed–Muller codes," *Journal of Computer and System Sciences*, vol. 72, pp. 786–812, 2006.

[48] D. Zuckerman, "Linear degree extractors and the inapproximability of Max Clique and Chromatic Number," *Theory of Computing*, vol. 3, pp. 103–128, 2007.

[49] L. Adleman, "Two theorems on random polynomial time," in *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1978)*. IEEE, 1978, pp. 75–83.

[50] D. Doron, D. Moshkovitz, J. Oh, and D. Zuckerman, "Nearly optimal pseudorandomness from hardness." in *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 26, 2019, p. 99.

[51] B. Barak, A. Rao, R. Shaltiel, and A. Wigderson, "2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl-Wilson construction," *Annals of Mathematics*, vol. 176, no. 3, pp. 1483–1544, 2012.

[52] N. Alon, J. Bruck, J. Naor, M. Naor, and R. M. Roth, "Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 509–516, 1992.

[53] N. Alon, J. Edmonds, and M. Luby, "Linear time erasure codes with nearly optimal recovery," in *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1995)*. IEEE, 1995, pp. 512–519.

[54] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf, "High-rate locally correctable and locally testable codes with sub-polynomial query complexity," *Journal of the ACM (JACM)*, vol. 64, no. 2, p. 11, 2017.

[55] I. Dinur, P. Harsha, T. Kaufman, I. L. Navon, and A. T. Shma, "List decoding with double samplers," in *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*. SIAM, 2019, pp. 2134–2153.