Extractors and pseudo-random generators with optimal seed length

Russell Impagliazzo[*]
Computer Science and Engineering
UC, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114
russell@cs.ucsd.edu

Ronen Shaltiel
Department of Computer Science
Hebrew University
Jerusalem, Israel
ronens@cs.huji.ac.il

Avi Wigderson[†]
Department of Computer Science
Hebrew University
Jerusalem, Israel
and
Institute for Advanced Study
Princeton
avi@cs.huji.ac.il

ABSTRACT

We give the first construction of a pseudo-random generator with optimal seed length that uses (essentially) arbitrary hardness. It builds on the novel recursive use of the NW-generator in [8], which produced many optimal generators one of which was pseudo-random. This is achieved in two stages - first significantly reducing the number of candidate generators, and then efficiently combining them into one.

We also give the first construction of an extractor with optimal seed length, that can handle sub-polynomial entropy levels. It builds on the fundamental connection between extractors and pseudo-random generators discovered by Trevisan [21], combined with construction above. Moreover, using Kolmogorov Complexity rather than circuit size in the analysis gives super-polynomial savings for our construction, and renders our extractors better than known for all entropy levels.

1. INTRODUCTION

A central question in Complexity Theory concerns the power of probabilistic algorithms. Such algorithms are allowed to use a string of independent coin tosses in their computation. Two different directions to this problem resulted in rich and elaborate theories.

The first direction (introduced by [3; 23]) tries to see if probabilistic algorithms can function if given many fewer random bits than they want to use. The idea is to use a deterministic procedure, called a *pseudo-random generator*, to stretch these few independent bits (the short *seed*) into the appropriate length. The distribution produced should "look random" to all efficient algorithms. The reader is referred to the excellent monograph [6] devoted to this field and its varied connections to complexity theory, cryptography and learning theory. But perhaps the most important connection is that given such a generator, a simple nontrivial deterministic simulation of the given probabilistic algorithm follows, which is exponential in the "seed" length. It is thus crucial to reduce this parameter.

The second direction (introduced by [2; 19] asks if such algorithms can function when their random input is not independent unbiased bits, but rather the output of some defective random source. Such a source contains sufficient entropy, but otherwise can be arbitrary. The idea is to use deterministic procedures, called *extractors*, to convert this "hidden" entropy and a small number of additional random bits, (called the *seed*) into a (nearly) uniform distribution. The reader is referred to the excellent survey papers [12; 13] on the many varied uses of extractors in complexity, combinatorics and network design. It can be shown that the need to use a seed of additional randomness is inevitable. However, enumerating all possibilities for this "seed" enables the extractor to use weak sources in probabilistic algorithms. Again, minimizing the length of this "seed" is crucial for the efficiency of this conversion.

A major result in the first direction was discovered in [14]. They showed that every difficult problem (in E) could be used to construct a pseudo-random generator. The quality of this NW-generator (i.e. its seed length) was shown to relate to the difficulty of the given function. Their work has been quantitatively improved and qualitatively extended ([1; 7; 9; 10; 20; 8; 11; 4]), but their construction remains central to work in this area. While the best "hardness vs. randomness" trade-off to be expected from such a construction should yield a seed whose size is `linear` in the input size of the given hard function, this was not achieved yet, and the best construction so far [20] has seeds that can be nearly `quadratic`.

In another major result, Trevisan ([21]) recently showed that the two directions above are intimately linked. He proved

[*]Research Supported by NSF Award CCR-9734911, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF Grant 97-00188

[†]This research was supported by grant number 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities

that any variant of the NW-generator (converting arbitrary hard functions into pseudo-random distributions) could be used to construct an extractor. Moreover, the "seeds" in both constructions are the same length under this translation. [21] (and then [18]) proceeded to use the NW-generator to give the best and simplest constructions of extractors known. As above, even the best can sometimes require nearly quadratic seed size in the "optimal" value (proved existentially by a counting argument matching the known lower bound). Thus, improving the NW-generator will impact both directions.

In [8], a new, more sophisticated way of using the NW-generator was proposed. The main idea was to use a recursive application of the NW construction, to construct either a pseudo-random generator with optimal seed, or a family of functions on much smaller inputs one of which had almost the same hardness as the original, (and harder than the original function when measuring hardness relative to input size). Thus, either an optimal seed pseudo-random generator is constructed, or a harder function is obtained. Which of these events occurred, and, if the latter, which of the "smaller" functions was hard, were unknown. Still, repeating this idea recursively was shown to terminate, resulting in exponentially many optimal seed generators, one of which was guaranteed to be pseudo-random. For the purpose of deterministic simulation of probabilistic algorithms [8] showed that the above suffice, and gave nearly tight trade-offs for hardness vs. randomness. However, the non-constructive element above did not yield one pseudo-random generator, and in particular could not be used for constructing extractors.

In this paper, we overcome this difficulty. We construct both extractors and pseudo-random generators that have optimal seed length using a recursive application of the NW-generator. It yields the same hardness vs. randomness trade-offs as in [8] directly, without resorting to the "generators tournament" forced by the multitude of generators there. Furthermore, the extractors we obtain from this construction are the first for sub-polynomial entropies to have optimal seed lengths. They also give small improvements over other constructions for all entropy levels.

The main new ideas are:

1. [8] show that if the NW generator fails, at least one of a family of restricted functions is almost as hard as the original one. We show that this is true for a non-negligible fraction of the family, and use this as the basis for a *probabilistic* construction of a *single* hard restricted function. We recurse with this function alone, to obtain only very few generators one of which is pseudo-random. These, being so few, we can afford to use them all with independent seeds and xor the outputs. Finally, we show that the number of random bits used in the probabilistic construction is small enough to be included in the seed.

2. The translation of the result above to the language of extractors gives constructions which extract only sub-polynomial part of the entropy in the source. This happens because every recursive call loses a polynomial fraction of the initial circuit complexity. This difficulty can be avoided when constructing extractors. It turns out that for this purpose one can replace the hardness measure, and use Kolmogorov complexity instead of circuit complexity. Doing so, every recursive call loses only a small amount of hardness, and as a result, we can extract any polynomial fraction of the initial entropy and at the same time simplify the argument.

Table 1: Pseudo-random generators comparison

Reference	seed size	circuit size
[1]	$d = O(l^4 \log^3 k)$	$m = k^{\Omega(1)}$
[7]	$d = O(l^2 \log k)$	$m = k^{\Omega(1)}$
[9]*	$d = O(\frac{l^4}{\log^3 k})$	$m = k^{\Omega(1)}$
[20]	$d = O(\frac{l^2}{\log k})$	$m = k^{\Omega(1)}$
Theorem 1	$d = O(l)$	$m = k^{\Omega(\frac{1}{\log logl})}$
Ultimate goal	$d = O(l)$	$m = k^{\Omega(1)}$

All results assume the existence of a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$, and that circuits of size $k(l)$ cannot compute.

* Impagliazzo and Wigderson state their result for $k(l) = 2^{\Omega(l)}$ and their result gives $d = O(l)$, (which implies $BPP = P$) for such an assumption.

We use these ideas to prove:

THEOREM 1 (**Optimal seed generators**). *For every function $k(l)$ with $l < k(l) < 2^l$, there is a function $m(l) = k(l)^{\Omega(1/\log\log l)}$ so that the following holds: Assume there is a function $f = \{f_l\}$ that is computable in time $2^{O(l)}$, so that for all l, f_l cannot be computed by circuits of size $k(l)$. Then there exists a generator $G : \{0,1\}^{O(l)} \to \{0,1\}^{m(l)}$ that is computable in time $2^{O(l)}$, and has the property that every circuit of size $m(l)$ is $1/m(l)$-fooled[1] by the distribution induced by G.*

This theorem is stated more precisely in theorem 5 and its proof is given in section 4. See table 1 for comparison with previous results.

THEOREM 2 (**Optimal seed extractors**). *For every n, k and ϵ there exists an explicit $(k, \frac{1}{m})$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{\epsilon})} \to \{0,1\}^m$, where $m = k^{1-\delta}$ for arbitrary $\delta > 0$.*

See table 2 for comparison with previous results. An interesting feature of our construction is that it seems to favor sources with low entropy, (whereas the extractors of [21; 18] are more efficient when the entropy is large). An example is the following theorem which asserts that we can get a large improvement in the number of extracted bits if the entropy is small.

[1] A distribution D on $\{0,1\}^m$, ϵ-fools a test $A : \{0,1\}^m \to \{0,1\}$ if

$$|Pr_{x \in_D \{0,1\}^m}[A(x) = 1] - Pr_{x \in_{U_m} \{0,1\}^m}[A(x) = 1]| \leq \epsilon$$

In the above theorem we identify between the circuit size and the input size. Without loss of generality we view all circuits of size m as circuits on m bits. If the circuit takes as input $j < m$ bits, we assume that it takes m bits and ignores the last $m - j$ bits.

THEOREM 3. *For every n, k and ϵ there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{\epsilon} + \log^2 k)} \rightarrow \{0,1\}^m$, for $m = \Omega(k/\log\log\log n)$.*

1.1 Organization of the paper

In section 2, we define pseudo-random generators and extractors, state our exact results, and present the necessary background definitions and results. In section 3, we present our main construction, and prove that it "transforms hardness into pseudo-randomness". In section 4, we show how to deduce our results from the main construction.

2. DEFINITIONS AND INGREDIENTS

2.1 Pseudo-randomness

We want to say that the outputs of our pseudo-random generators and extractors look like random strings as far as relevant tests are concerned. We formalize this below, with A standing for the test. Let U_m be the uniform distribution on m bit strings, and let $x \in_D \{0,1\}^m$ mean that x is selected from all m bit strings according to probability distribution D. For a function G on d bits, we use G to denote the distribution $G(U_d)$.

DEFINITION 1. *Let A be a boolean predicate on m bit strings. We say that a distribution D on m bit strings* ϵ-fools *A if $|Prob_{x \in_D \{0,1\}^m}[A(x)] - Prob_{x \in_{U_m} \{0,1\}^m}[A(x)]| < \epsilon$.*

For a class $\mathcal{A}$ of such predicates, D is ϵ-pseudo-random *for $\mathcal{A}$ if D ϵ-fools each $A \in \mathcal{A}$. We'll be particularly interested in the following classes of predicates:*

- *The class of all predicates (on m bits): D is called* ϵ-uniform *if it ϵ-fools this class*[2].
- *The class $Size_m$ of all predicates computable by circuits with m or fewer gates: D is called* (ϵ, m)-pseudo-random *if it ϵ-fools this class. (We use the same parameter for the circuit size and the number of inputs, as a circuit can always ignore some of it's inputs.)*

  For functions $k(l), d(l), m(l), \epsilon(l)$, a (k, ϵ)-pseudo-random generator is a (family) of functions $G_l : \{0,1\}^{d(l)} \rightarrow \{0,1\}^{m(l)}$, that is computable in time $2^{O(d)}$,[3] *such that the distribution $G(U_{d(l)})$ (of outputs of the generator) is $(k(l), \epsilon(l))$-pseudo-random.*

2.2 Complexity measures for functions and strings

We identify between functions $f : \{0,1\}^l \rightarrow \{0,1\}$ and strings in $\{0,1\}^{n=2^l}$ in the obvious way setting $f_i = f(i)$. This identification is helpful since we use two complexity measures: circuit complexity (which is defined for functions and is the measure we use for constructing pseudo-random generators) and Kolmogorov complexity (which is defined for strings and is the measure we use for constructing extractors). For both measures, we will argue that our construction transforms a "hard" function/string into a "pseudo-random" distribution. The exact meaning of this statement is that if a test A is not fooled by the "pseudo-random" distribution then A can be used as an oracle to compute/describe the initial function/string. Thus, we define circuit complexity and Kolmogorov complexity relative to a predicate A.

DEFINITION 2. *Let A be a predicate on m bit inputs.*

- *Define $S_A(f)$ to be the size of the smallest circuit that computes f and is allowed to use A-gates, (in addition to the standard boolean gates). Thus, $S(f)$ denotes the circuit complexity of f. We use the same convention in the next definitions.*
- *For $0 < \delta < 1/2$, let $S_{A,\delta}(f)$ be the minimum of $S_A(f')$ over all strings f' with hamming distance at most $\delta|f|$ from f.*

The last item in the above definition measures circuit complexity of functions on random inputs. If $S_{A,\delta}(f)$ is large, then f is not only hard to compute, but also hard to compute on average. We proceed and define these concepts for Kolmogorov complexity.

DEFINITION 3. *Let A be a predicate on m bit inputs.*

- *Define $K_A(f)$, the Kolmogorov complexity of f given A, as the length (in bits) of the smallest description of f which is allowed to use a description of A for free, i.e., that of an oracle Turing Machine which, using oracle A, outputs f.*
- *For $0 < \delta < 1/2$, let $K_{A,\delta}(f)$ be the minimum of $K_A(f')$ over all strings f' with hamming distance at most $\delta|f|$ from f.*

2.3 Hardness versus Randomness

The only known constructions of pseudo-random generators use unproven assumptions. The most common assumption used is the existence of "hard functions". Specifically one assumes that there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$, yet every circuit of size $k(l)$, (where k is an integer function which measures the hardness of f), cannot compute f. Using this assumption a large number of papers [1; 7; 9; 10; 20; 11; 4] construct Pseudo-random generators. Most of these results use the Nisan Wigderson generator exactly as stated in [14], and improvements are gained by a "pre-processing stage of hardness amplification".
In some sense, (which is made exact in [8]) the best possible pseudo-random generator expected by such a transformation is one which uses $O(l)$ random bits to fool circuits of size $k(l)$.
The best previously known construction is by [20].

THEOREM 4. *[20] If there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, $S(f_l) > k(l)$, then there exists a function $m(l) = k(l)^{\Omega(1)}$ and a $(m(l), \frac{1}{m(l)})$-pseudo-random generator with seed length $O(\frac{l^2}{\log k(l)})$.*

In this paper we get the optimal seed length. A small gap from the optimal expected generator is left since we are only able to fool slightly smaller circuits[4].

[2] It is easy to verify that a distribution D is ϵ-uniform if and only if the statistical (L1-norm) distance between D and U_m is bounded by $\epsilon/2$.

[3] Pseudo-random generators are allowed to run in exponential time. This is because when using them to derandomize a probabilistic algorithm they are run an exponential number of times for all their possible seeds.

[4] The following theorem rephrases theorem 1 with our notation

Table 2: Extractors comparison

reference	min-entropy k	output length m	additional randomness d
[21]	any k	$m = k^{1-\delta}$	$d = O(\frac{\log^2 n}{\log k})$
Theorem 2	any k	$m = k^{1-\delta}$	$d = O(\log n)$
[18]	any k	$m = (1-\delta)k$	$d = O(\log^2 n)$
Theorem 3	any k	$m = \Omega(\frac{k}{\log\log\log n})$	$d = O(\log n) + O(\log^2 k)$
Theorem 3 + [22]	$k \leq 2^{\sqrt{\log n}}$	$m = (1-\delta)k$	$d = O(\log n \cdot \log\log\log n)$
Ultimate goal	any k	$m = k$	$d = O(\log n)$

The results are given for constant ϵ.
δ is an arbitrary small constant.

THEOREM 5. *If there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, $S(f_l) > k(l)$, then there exists a function $m(l) = k(l)^{\Omega(\frac{1}{\log\log l})}$ and a $(m(l), \frac{1}{m(l)})$-pseudo-random generator with seed length $O(l)$.*

This does not improve derandomization of probabilistic algorithm as such hardness vs. randomness tradeoffs were already achieved in [8], without constructing a pseudo-random generator.

REMARK 1. *Trevisan [21], observed that all hardness vs. randomness tradeoffs use no consequences of the assumption that f is computable in time $2^{O(l)}$ but the trivial one: that the generator can compute the function. Thus, all these constructions can be seen as taking two inputs: the function f, (encoded as an $n = 2^l$ bits truth table), and the seed. We call such transformations pseudo-random generator schemes.*

2.4 Extractors

Roughly speaking, an extractor is a function which uses a small amount of random bits to extract a lot of (almost) truly random bits from a weak random source which contains sufficient randomness. The following variant of the entropy function is used to quantify the amount of randomness which a distribution contains.

DEFINITION 4. *A distribution P on $\{0,1\}^n$ is said to have min-entropy (at least) k, if for every $x \in \{0,1\}^n$, $P(x) \leq 2^{-k}$.*

DEFINITION 5. *A function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is called a (k, ϵ)-extractor if for every distribution P on $\{0,1\}^n$ which has min-entropy k, the distribution $Ext(P, U_d)$ is ϵ-uniform*[5]*. We say that a family of extractors is explicit if it can be computed in polynomial time.*[6]

Our motivation in this paper is constructing extractors with small d, (for given n,k and ϵ). A lower bound of $d = \Omega(\log(n/\epsilon))$ is given by [15; 16].

[5]Let us remark again that a distribution is ϵ-uniform if and only if it has statistical distance at most $\epsilon/2$ from the uniform distribution. Thus this definition coincides with the standard one

[6]Formally, a family $E = \{E_n\}$ of extractors is defined given polynomially computable integer functions $d(n), m(n), k(n), \epsilon(n)$ where $E_n : \{0,1\}^n \times \{0,1\}^{d(n)} \to \{0,1\}^{m(n)}$ is a $(k(n), \epsilon(n))$ extractor. The family is explicit in the sense that E_n can be computed in time polynomial in n.

DEFINITION 6. *Let $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ and let $h \in \{0,1\}^n$. Then we define $Ext^h : \{0,1\}^d \to \{0,1\}^m$ by $Ext^h(x) = Ext(h, x)$, and call it* the generator derived from *Ext* by h.

The following theorem, (which is the main observation of [21]) asserts that pseudo-random generator schemes can be used to construct extractors. Interestingly, for this transformation it is more natural to measure hardness by Kolmogorov complexity than by circuit complexity.

THEOREM 6. *[21] Let $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a function, Assume, for any predicate A, and any $h \in \{0,1\}^n$, if Ext^h does not ϵ-fool A, then $K_A(h) \leq k$. Then Ext is a $(k + \log(1/\epsilon), 2\epsilon)$-extractor.*

PROOF. Let A be any test, and let P be a distribution with min-entropy at least $k + \log(1/\epsilon)$. The bias of A on $Ext(P, U_d)$ is the expectation for $h \in_P \{0,1\}^n$ of the bias of Ext^h. Now, for all but 2^k h's, the latter bias has absolute value less than ϵ. The at most 2^k exceptions have total probability at most $2^k/2^{min-entropy(P)} \leq \epsilon$. Therefore, the total bias is at most 2ϵ. $\square$

There is also a partial converse:

THEOREM 7. *Let $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be an explicit (k, ϵ)-extractor. Then, for any predicate A, and any $h \in \{0,1\}^n$, if Ext^h does not ϵ-fool A, then $K_A(h) \leq k + O(1)$.*

PROOF. Let H_A be the set of all h so that Ext^h does not ϵ-fool A. H_A is constructible using A as an oracle, so $K_A(h) \leq \log|H_A| + O(1)$ for any $h \in H_A$. Without loss of generality, assume that, for half the elements h in H_A, A is ϵ more likely for an output of Ext^h than for a random element. Then the same is the case if h is chosen uniformly from this subset of H_A, and x is chosen uniformly, and we compute $A(Ext(h, x))$. Thus, by the definition of extractors, this distribution on A has min-entropy less than k , i.e., $|H_A|/2 < 2^k$ or $\log|H_A| < k + 1$. $\square$

Following Trevisan, we use theorem 6 to reduce the problem of constructing extractors into proving hardness versus randomness tradeoffs. We use theorem 7 to place known explicit extractors at the base of our recursion.

The following two theorems summarize the best known explicit extractors for general min-entropy.

THEOREM 8. *[21] For every n, k there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\frac{log^2 n}{\log k})} \rightarrow \{0,1\}^m$ (where $m = k^{1-\delta}$ for arbitrary $\delta > 0$).*

THEOREM 9. *[18] For every n, k there exists an explicit $(k, \frac{1}{m})$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log^2 n)} \rightarrow \{0,1\}^m$ (where $m = (1-\delta)k$ for arbitrary $\delta > 0$).*

In this paper we construct the following explicit extractors.

THEOREM 10. *For every n, k there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n)} \rightarrow \{0,1\}^m$, (where $m = k^{1-\delta}$ for arbitrary $\delta > 0$).*

THEOREM 11. *For every n, k there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n + \log^2 k)} \rightarrow \{0,1\}^m$, for $m = \Omega(k/\log\log\log n)$.*

Note that this extractor uses optimal seed length for $k < 2^{\sqrt{\log n}}$. In [22] it was shown how to transform an extractor which extract $m = k/r$ bits into one which extracts $m = (1-\delta)k$, (for arbitrary constant δ) while multiplying d by $O(r)$. Using this with theorem 11 gives:

THEOREM 12. *For every n and $k < 2^{\sqrt{\log n}}$ there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n \cdot \log\log\log n)} \rightarrow \{0,1\}^m$, (where $m = (1-\delta)k$ for arbitrary $\delta > 0$).*

Note that we improve Trevisan's extractor and use optimal ($d = O(\log n)$) seed length for extracting $k^{1-\delta}$ bits. For low min-entropy (i.e. $k \leq 2^{\sqrt{\log n}}$) we extract a larger fraction of the entropy. This larger fraction is still small compared to that of [18]. However, the advantage of our construction is that it maintains optimal seed length. If extracting a constant fraction of the min-entropy is desired we also substantially improve [18] for low min-entropy[7].

Our technique yields extractors with rather large error, ($\epsilon = 1/m$). In [17] it was shown how to transform an extractor with large error into one that works for any error. When one starts with error $\epsilon = 1/m$, this transformation increases the seed only by the inevitable and optimal $O(\log \frac{1}{\epsilon})$ while preserving the other parameters. (The reader is referred to [17] for exact formulation). Performing this transformation gives theorems 2,3.

2.5 The Nisan-Wigderson generator

Almost all schemes of converting hardness into pseudo randomness, as well as the above extractor constructions, use the NW-generator from [14]. Their construction converts a "hard" Boolean function f on l bit inputs, into a pseudo-random generator taking an input seed of size $d > l$ to an output of length $m >> d$. To use the construction for derandomization, one needs to specify the hard function f, and a family Δ of subsets of $\{1, \cdots, d\}$ such that each pair of sets has small intersection. Such families are called "designs", and the intersection sizes determine the quality of the pseudo-random generator.

[7] Actually, our results slightly improve those of [18] and [21] for all k, as all terms of the form $\log\log\log n$ can be replaced by $\log\log \frac{4\log n}{\log k}$.

DEFINITION 7. *A family of sets $\Delta = (S_1, \cdots, S_m \subseteq [d])$ is called a (l,u)-design if*

- *For all i, $|S_i| = l$.*
- *For all $i \neq j$, $|S_i \cap S_j| \leq u$.*

DEFINITION 8. *(The NW-generator) Let $l < d < m$ be integers, and let $n = 2^l$. Let Δ be a (l,u)-design. Define a function $NW^\Delta : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ in the following way: Given $f \in \{0,1\}^n$ and $x \in \{0,1\}^d$, we view f as a function over $l = \log n$ bits, (by having $f(v) = f_v$). Let $x|_S$ denote the $|S|$ bit string obtained by restricting x to the indices in S. Define:*

$$NW^\Delta(f,x) = f(x|_{S_1}) \circ f(x|_{S_2}).. \circ f(x|_{S_m})$$

For a fixed f, let $NW^{f,\Delta}$ be the function from $\{0,1\}^d \rightarrow \{0,1\}^m$ defined by $NW^{f,\Delta}(x) = NW^\Delta(f,x)$.

Given Δ, f and x as inputs, $NW^\Delta(f,x)$ can be computed in polynomial time. $NW^{f,\Delta}(x)$ can be computed, given Δ, in time polynomial in m, with an oracle for f. [14] also show how to construct good designs Δ.

THEOREM 13. *[14] There exist constants c_1, c_2 such that for every l, d, m, such that $l < d < m$ and $d > c_2 \log m$ there exists a (l,u)-design $\Delta = (S_1, \cdots, S_m \subseteq [d])$, with $u = max(\frac{c_1 l^2}{d}, \log m)$, Furthermore, this design can be constructed in time $poly(2^d)$.*

The proof that the NW construction is a good pseudo-random generator involves looking at certain restrictions of f.

DEFINITION 9. *Given an (l,u)-design $\Delta = (S_1, \cdots, S_m \subseteq [d])$ and a function $f : \{0,1\}^l \rightarrow \{0,1\}$, define a collection of functions $\mathcal{R}_f^\Delta = \{f_{i,j,\beta}^\Delta | \beta \in \{0,1\}^d, 1 \leq i < j \leq m\}$ as follows:*

$$f_{i,j,\beta}^\Delta : \{0,1\}^u \rightarrow \{0,1\}$$

is the function defined by: On input $z \in \{0,1\}^u$, construct a string $s \in \{0,1\}^d$ by first assigning, for each $p \notin S_i \cap S_j$, $s_p = \beta_p$, and then filling the remaining (at most) u bits according to (the first bits of) z.

$$f_{i,j,\beta}^\Delta(z) = NW^{f,\Delta}(s)_j$$

For $1 \leq i \leq m$, let $f_{i,\beta}^\Delta(j,z) = f_{i,j,\beta}^\Delta(z)$. We refer to these functions as the "restrictions" of f. Note that the input size of each such restriction is $u + \log m$.

Note that given Δ and a function f (encoded as a $n = 2^l$ bit truth table) and $\beta \in \{0,1\}^d$ the truth table of $f_{i,\beta}^\Delta$ can be computed in polynomial time. These functions are over $u + \log m$ bits, which trivially entails that $S(f_{i,\beta}^\Delta)$ and $K(f_{i,\beta}^\Delta)$ are bounded by $m2^u$.
Implicit in [14] is the following[8]:

[8] [14] prove that $NW^{f,\Delta}$ is ϵ-pseudo-random for all tests computable in size $S_{1/2-\Omega(\epsilon/m)}(f)/(m2^u)$. This means that the existence of functions which are "hard to approximate" implies the existence of a pseudo-random generator.
With the above terminology, the argument of [14] can be presented this way: [14] uses designs with very small u, (which in turn forces d to be relatively large). This makes the circuit complexity of all the $f_{i,\beta}^\Delta$'s relatively small. Lemma

LEMMA 1. *There is a polynomial-time oracle Turing Machine $M^{g,A}$ with the following property. Assume $NW^{f,\Delta}$ does not ϵ-fool a test A. Choose uniformly a β, an i, an m bit string α, and an l bit string x. Then if M is run on (x, α, β) using oracles $g = f^{\Delta}_{i,\beta}$ and A, $Prob[M^{g,A}(x, \alpha, \beta) \neq f(x)] < 1/2 - \Omega(\epsilon/m)$.*

For completeness we prove lemma 1 in the appendix. This view of the NW-construction is the key observation of [8]. The above lemma is then used to connect between the complexity of the original function, that of the restricted functions, and the power of the generator.

COROLLARY 1. *[8] If $NW^{f,\Delta}$ does not ϵ-fool A, then there are β and i so that: $S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$.*

Corollary 1 gives that if $NW^{f,\Delta}$ does not fool A, then one of the restrictions can be used to show that f is "easy", (at least in the sense that there exists a small circuit that uses A gates and "approximates" f). This is useful since all the restrictions are on only $u + \log m$ bits. Thus, if we start with a function f that is "hard" for A, either the $NW^{f,\Delta}$ fools A or we obtain a "smaller" function that is almost as hard as the original one. This observation was used recursively in [8]. The problem was that in the case that A is not fooled only *one* of the restrictions could be shown to be hard. This causes [8] to use a costly exhaustive search to "find" the hard restriction.
In this paper we observe of that the above result can be strengthened to show that in the case that A is not fooled by $NW^{f,\Delta}$ a non-negligible fraction of the restrictions are hard. Intuitively, this will enable us to select a small number of these restrictions using a randomized procedure such that with high probability we will obtain a hard restriction.

COROLLARY 2. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$

Corollary 2 follows just as easily from lemma 1 using a standard argument. An analogous argument can be used to state corollary 2 for Kolmogorov Complexity. It turns out that by considering Kolmogorov complexity the same argument gives much better parameters! This happens because in the Kolmogorov complexity setting, the running time of M doesn't count. Thus, we get an additive term and not a multiplicative term.

COROLLARY 3. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$K_{A,1/2-\Omega(\epsilon/m)}(f) \leq K_A(f^{\Delta}_{i,\beta}) + 2m$

2.6 Xoring generators

We will recursively construct a sequence of pseudo-random generators/extractors, one of which will fool the relevant predicate A. We will need a way of combining them. Fortunately, unlike in [8], the number of such generators will be small. We'd like to "xor" all the generator's into one generator. A technical inconvenience is that the "xor generator" does not necessarily fool A. The following definition and trivial lemma enable us to overcome this difficulty.

DEFINITION 10. *For a predicate A on m bits and $y \in \{0,1\}^m$, define a predicate $A^{\oplus y}$ on m bits by having $A^{\oplus y}(x) = A(x \oplus y)$. Define $A^{\oplus}$ to be the class of all predicates $A^{\oplus y}$.*

LEMMA 2. *Let $P_1, \cdots, P_r$ be distributions on $\{0,1\}^m$ and $A : \{0,1\}^m \rightarrow \{0,1\}$ be a predicate. Suppose that one of the P_i's is ϵ-pseudo-random for $A^{\oplus}$. Consider the distribution $\bar{P} = P_1 \oplus \cdots \oplus P_r$, which samples independently $z_i \in_{P_i} \{0,1\}^m$, and outputs $z_1 \oplus \cdots \oplus z_r$. Then the distribution $\bar{P}$ ϵ-fools A.*

The price of lemma 2, is that if you have two candidate generators. $G_1 : \{0,1\}^{d_1} \rightarrow \{0,1\}^m$, $G_2 : \{0,1\}^{d_2} \rightarrow \{0,1\}^m$ the $\oplus$-generator $G(x_1, x_2) = G_1(x_1) \oplus G_2(x_2)$ takes a seed of length $d_1 + d_2$. This means that "xoring" many generators blows up the seed length. We want to only increase the seed length of a single generator linearly. We will be able to avoid increasing the total seed length by more than a constant factor over that of G_1, by making sure that the seed lengths are decreasing exponentially. In particular, this means that there can be only logarithmically many (in the seed length) elements in the sequence, a contrast with [8], where the recursive construction created an exponential size (in the seed length) family of generators.
Let us rephrase corollaries 2, 3, and replace A by $A^{\oplus}$. This change does not affect the parameters by much. To convert a circuit using $A^{\oplus y}$ gates to one using A gates, we can replace the $A^{\oplus y}$ gates with A gates, and negate wires going to the i'th input of an $A^{\oplus y}$ gate if $y_i = 1$. This gives:

COROLLARY 4. *If $NW^{f,\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$

As to Kolmogorov complexity, note that by giving y, we can convert a machine with oracle access to $A^{\oplus y}$ into one with oracle access to A.

COROLLARY 5. *If $NW^{f,\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$K_{A,1/2-\Omega(\epsilon/m)}(f) \leq K_A(f^{\Delta}_{i,\beta}) + 4m$

2.7 Hardness Amplification

The above connections relate the quality of the generator and the complexity of the specified restrictions to the complexity of *approximating* the function f, i.e., computing a function f' that has non-negligible correlation to f. Much of the work on improving the results in [14] concerns constructing a hard to approximate function from one that is hard to compute in the worst-case ([1; 7; 9; 20]). This process is usually called hardness amplification. Here, we'll use the hardness amplification from [20], which is nearly optimal.

THEOREM 14. *[20] There exists a polynomial time algorithm that given a function $f : \{0,1\}^l \rightarrow \{0,1\}$ (encoded as a 2^l bit truth table) and ρ, produces the truth table of a function $\hat{f} : \{0,1\}^{4l} \rightarrow \{0,1\}$, with the following properties for any predicate A:*

1 shows that any circuit A of size m which is not fooled by $NW^{f,\Delta}$ can be combined with the circuits for the restricted functions to construct a circuit of size $poly(m)2^u$ which approximates f. Thus, if f is assumed to be hard to approximate by such circuits, the distribution induced by the generator is pseudo-random for $Size_m$.

1. $S_A(f) \le S_{A,1/2-\rho}(\hat{f})(\frac{l}{\rho})^{O(1)}$

2. $K_A(f) \le K_{A,1/2-\rho}(\hat{f}) + O(\log \frac{1}{\rho})$

Combining this with the corollaries from the last section gives the following analogous results for circuit complexity and Kolmogorov complexity. Note that again the same argument produces more efficient parameters in the Kolmogorov complexity setting[9].

Corollary 6. *If $NW^{\hat{f},\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$S_A(f) \le S_A(\hat{f}^{\Delta}_{i,\beta})(m/\epsilon)^{O(1)}$

Corollary 7. *If $NW^{\hat{f},\Delta}$ is not ϵ-pseudo-random $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i :*
$K_A(f) \le K_A(\hat{f}^{\Delta}_{i,\beta}) + 4m + O(\log \frac{1}{\epsilon})$

The above lemmas give the same intuitive connection between the hardness of the function f and its restrictions. The difference is that now we measure "worst-case" hardness in both sides of the inequality. This enables us to use these corollaries recursively.

3. OUR CONSTRUCTION

In our construction, we use the techniques from the previous sections in a recursive way, as do [8]. The idea is that if the NW generator fails on $\hat{f}$, then *many* of the restrictions are almost as complex as f was, and much smaller than f. In [8], all such restrictions were recursively converted into pseudo-random generators. Instead, we give a randomized procedure which constructs a single smaller function, by combining many randomly sampled restrictions. Since, the only way our construction can get random bits to perform this reduction is by including them in the seed, we use pairwise independence to minimize the number of bits actually used. At the end, we will have a relatively small number of generators, (one for each level of the recursion), and we combine them using the $\oplus$ construction of section 2.6.

We use the same construction for both extractors and pseudo-random generators, using our standard convention that boolean functions over l bits are equivalent to strings of length $n = 2^l$. Our construction is given as input the following ingredients:

- The length of strings in the source distribution n. (Or alternatively for pseudo-random generators $n = 2^l$ where l is the number of inputs of the hard function).
- The length of the output m.
- An explicit extractor (Pseudo-random generator scheme), $Base : \{0,1\}^{m^{20}} \times \{0,1\}^{d'} \to \{0,1\}^m$. To be used as the base of the recursion. (We will soon connect the quality of $Base$ to that of our construction).

[9] [20] is concerned only with circuit complexity. Still, their work gives the cited result for Kolmogorov complexity, and it turns out that the proof is even simpler for this case. Intuitively, this happens for the same reasons as before. loosely speaking, [20] construct a polynomial time Turing machine which computes f given a function that approximates $\hat{f}$. As before, the circuit complexity bound involves the time of this machine, whereas the Kolmogorov complexity bound only needs that a Turing machine is of constant size.

We construct a function $Rec : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$. ($d$ will soon be determined). We now describe the computation of Rec:

It begins by computing three sequences of integers $l_1, \cdots, l_r$, $u_1, \cdots, u_r$, $d_1, \cdots, d_r$ recursively as follows:

1. $d_1 = c_3 \log n$, where c_3 is a constant to be fixed later.
2. $d_{t+1} = \frac{d_t}{2}$.
3. $l_1 = 4 \log n$.
4. $l_{t+1} = 4(u_t + 5 \log m)$.
5. $u_t = max(\frac{c_1 (l_t)^2}{d_t}, \log m)$
6. r is such that $u_{r-1} = \log m$, (We will soon prove that such an r exists).

We view the first input of Rec as a boolean function f over l bits. It will be convenient to view the second input as composed of three parts: The first is used to obtain seeds for "candidate generators". We denote this part by $s = (s_1, \cdots, s_{r-1})$ where $s_t \in \{0,1\}^{d_t}$. (Note that the total length of s is bounded by $\sum d_t \le 2d_1$. The second part which we denote by s' will be a d' bits string which is used as a seed for $Base$. The third part which is of length $8d_1$ will allow Rec to perform randomized computations. Thus, in this presentation, we allow Rec to use randomization as long as the number of random bits used does not exceed $8d_1$. This makes the seed length $d = 2d_1 + d' + 8d_1 = O(\log n) + d'$.

For $1 \le t \le r$, Rec constructs a (l_t, u_t)-design $\Delta_t = (S_1, \cdots, S_m)$ with $S_i \subseteq [d_t]$, using theorem 13.

Rec then constructs truth tables for r functions $f_1, \cdots, f_r$ defined as follows: $f_1 = f$. For $t = 2$ to r, we pick m^4 pairs $(i_q, \beta_q) \in [m] \times \{0,1\}^{d_t}$ in a pairwise independent way. This requires $2(d_t + \log m) \le 4d_t$ random bits using the method from [5]. Thus, the total number of random bits used by Rec is no more than $4\sum_t d_t \le 8d_1$ as required.

Define f_{t+1} by:

$$f_{t+1}(q,j,z) = (\hat{f}_t)^{\Delta_t}_{(i_q,\beta_q)}(j,z)$$

Note that the truth table of each f_t can be computed in polynomial time given the truth table of f_{t-1}, and that the size of inputs to f_t are $u_{t-1} + 5 \log m$ bits. Thus, inputs to $\hat{f}_t$ are 4 times this length, or length l_t, as required.

It then sets $z_t = NW^{\hat{f}_t, \Delta_t}(s_t)$ for $t = 1, \cdots, r-1$, and $z_r = Base^{f_r}(s_r)$.

Finally, it outputs $z_1 \oplus z_2 \oplus ... \oplus z_r$.

The following lemmas assert that the construction produces extractors/pseudo random generators.

Lemma 3. *For every n and m, the recursion computing l_t, u_t, d_t terminates, and when it does $r \le \log\log \frac{4 \log n}{\log m} + 1$, and $l_r = 24 \log m$.*

Proof. We'll count the number of steps in which u_t decreases, until it reaches the limitation of $u_t \ge \log m$. For $i < r-1$, $u_t = \frac{c_1 (l_t)^2}{d_t}$. Using this rule, and the fact that $u_t \ge \log m$, we get that $u_t \le (\frac{c_1}{c_3})^{2^{t-1}} 11^{a_t} 2^{b_t} 4 \log n$, Where the sequences $\{a_i\}$ and $\{b_i\}$ are defined by the following recurrences: $a_1 = 2$, $a_{t+1} = 2(a_t + 1)$, $b_1 = 0$,

$b_{t+1} = 2b_t + t - 1$. One can compute that $a_t = O(2^{t-1})$, and $b_t = O(2^{t-1})$. Substituting this in the above equation we get that $u_t < (\frac{c_1 d}{c_3})^{2^{t-1}} \cdot 4 \log n$, for some constant d. Choosing $c_3 = 2dc_1$, we get that: $u_t < 2^{-2^{t-1}} 4 \log n$. This gives that $r \leq \log\log \frac{4 \log n}{\log m} + 1$. Note that since $u_{r-1} = \log m$, $l_r = 24 \log m$ $\square$

This implies that the construction stops after a small number of recursive steps, and can be computed in time polynomial in n, (which is exponential in l). We now show that this type of recursive construction produces extractors and pseudo-random generators.

Lemma 4. *There exists some constant c such that for every k', if Base is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, then Rec is an explicit $(k' + crm, 4/m)$-extractor $Rec : \{0,1\}^n \times \{0,1\}^{O(\log n)+d'} \to \{0,1\}^m$.*

We use the Kolmogorov complexity characterization of extractors. Lemma 4 follows from the following lemma and theorem 6.

Lemma 5. *For some constant c, if $K_A(f) > k' + crm$, then Rec^f $2/m$-fools A.*

Proof. We first need:

Lemma 6. *Let $1 \leq t \leq r-1$. With probability at least $1 - 1/m^2$ (over the choice of the m^4 pairs (i_q, β_q)), one of the following two events occur:*

- E_t^1: *$NW^{\hat{f}_t, \Delta_t}$ is $1/m^2$-pseudo-random for $A^{\oplus}$.*
- E_t^2: *$K_A(f_{t+1}) \geq K_A(f_t) - O(m)$.*

Proof. If E_t^1 does not occur, then by corollary 7, for a random (i, β), $K_A(f_t) \leq K_A((\hat{f}_t)^{\Delta_t}_{(i,\beta)}) + 5m$ with probability at least $\Omega(1/m^2)$. Let B be the set of such pairs. Recall that f_{t+1} "contains" each $(\hat{f}_t)^{\Delta_t}_{(i_q,\beta_q)}$ for $1 \leq q \leq m^4$. Therefore, if any $(i_q, \beta_q) \in B$, then $K_A(f_t) \leq K_A(f_{t+1}) + O(m)$. For each q, there is a chance of $\Omega(1/m^2)$ that $(i_q, \beta_q) \in B$. Having chosen m^4 such pairs we expect $\Omega(m^2)$ of them to fall in B. Since these events are pairwise independent we can use Chebyshev's inequality and get that the probability that all the pairs (i_q, β_q) "miss" B, is $O(1/m^2)$. $\square$

We define $E^1 = \cup_{1 \leq t \leq r-1} E_t^1$ and $E^2 = \cap_{1 \leq t \leq r-1} E_t^2$. Note that for all $1 \leq t \leq r-1$ the events E_t^1 and E_t^2 in the above lemma are determined before the seeds to the NW-generators are chosen. By the above lemma the probability that non of the events E^1,E^2 occurred, is bounded by $O(r/m^2) < 1/m$. If E^1 occurs then the conditional bias of Rec^f for A is at most $1/m$ by Lemma 2. If E^2 occurs then we can connect the hardness of f_t and f_{t+1} for all $1 \leq t \leq r-1$, and get that f_r is hard. Specifically, there exists a constant c such that $K_A(f_r) \geq K_A(f_1) - c(r-1)m \geq k' + 2m$. Using theorem 7 we get that when E^2 occurs, $Base^{f_r}$ $1/m$-fools A. (Verify that by lemma 3 $l_r = 24 \log m$, so f_r is an appropriate input for $Base$). However, in order to use lemma 2, we need to prove that $Base^{f_r}$ is $1/m$-pseudo-random for $A^{\oplus}$. This follows just the same since for every $y \in \{0,1\}^m$, $K_{A \oplus y}(f_r) \geq K_A(f_r) - m - O(1)$. By lemma 2 we get that if E^2 occurred then the conditional bias of Rec^f on A is at most $1/m$. This makes the total bias of Rec^f on A at most $1/m$ plus the probability that non of the events E^1,E^2 occurred, and the lemma follows.

The same argument gives that Rec is a pseudo-random generator scheme.

Lemma 7. *There exists some constant c such that for every k': Assume $Base^g$, (which has seed length d') is $(m, 1/m)$-pseudo-random, whenever a function $g : \{0,1\}^{24 \log m} \to \{0,1\}$ has $S(g) \geq k'$, then Rec^f, (which has seed length $d' + O(l)$) is $(m, 4/m)$-pseudo random, whenever a function $f : \{0,1\}^l \to \{0,1\}$ has $S(f) \geq m^{cr} k'$.*

The only difference in the proof is that one should use corollary 6 instead of 7.

4. RESULTS

In this section we show how "plugging" different bases to our construction gives the previously stated theorems. Theorem 10 follows using Trevisan extractor as $Base$.

Proof. (of theorem 10) We use the extractor of [21] (theorem 8) as base. Given a constant δ, we pick $k' = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}}$. Indeed, the extractor of [21] is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, where $d' = O(\log m)$. Using theorem 4 we get that Rec is an explicit $(k, 4/m)$-extractor, for $k = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}} + crm < m^{\frac{1}{1-\delta}}$. The seed length of Rec is $O(\log n) + d' = O(\log n)$. $\square$

Theorem 11 follows using the extractor of [18] as $Base$.

Proof. (of theorem 11) We use the extractor of [18] (theorem 8) as $Base$. We pick $k' = 2m$. Indeed, the extractor of [18] is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, where $d' = O(\log^2 m)$. Using theorem 4 we get that Rec is an explicit $(k, 4/m)$-extractor, for $k = 2m + O(rm) = O(m \log\log\log n)$ as required. The seed length of Rec is $O(\log n) + d' = O(\log n + \log^2 k)$. $\square$

The generator of theorems 1, 5 is achieved by Using the generator of [20] as $Base$.

Proof. (of theorems 1, 5) We use the pseudo-random generator scheme of [20] (theorem 4) as $Base$. We choose $k' = m^d$, (where d is the constant "hidden" in the Omega notation of theorem 4). Indeed, the generator of [20] is $(m, 1/m)$-pseudo-random when given a function $g : \{0,1\}^{24 \log m} \to \{0,1\}$ which cannot be computed by circuits of size k'. The seed length of the generator is $d' = O(l)$. Using theorem 7 we get that Rec^f is $(m, 4/m)$-pseudo-random when given a function $f : \{0,1\}^l \to \{0,1\}$ which cannot be computed by circuits of size

$$k = m^d m^{cr} = m^{O(\log\log l)}$$

The seed length of Rec is $O(l) + d' = O(l)$. $\square$

5. CONCLUSIONS AND OPEN PROBLEMS

We have shown how to improve the Nisan-Wigderson generator, to use optimal seed length for arbitrary hardness. However, there still remains a small gap between the output size of our pseudo-random generator and that of the best expected generator. (Namely, we get $m = k^{\Omega(\frac{1}{\log\log l})}$ whereas we expect to get $m = k^{\Omega(1)}$).

The same gap occurs in extractors, here the gap is quantitively smaller (since using Kolmogorov complexity we get $m = k^{1-\delta}$). However, the optimal extractor achieves $m = k$. Our construction works by reducing the problem of constructing extractors for general sources to that of constructing extractors for sources with polynomial min-entropy. Thus, any improvement of such extractors immediately implies an improvement for all entropy levels.

It is interesting to note that both our results and those of [18] follow from improving the Nisan-Wigderson generator, and then applying the Trevisan reduction. A natural question is: can both of these improvements be made simultaneously? A positive answer may give an extractor that gets the benefits of both constructions, (one that uses the optimal seed length to extract $m = (1-\delta)k$ bits).

Finally, an interesting feature of our construction is that, using the currently known extractors, this reduction actually improves the parameters for low min-entropy. As a result we have that extracting randomness from sources with low ($2^{\sqrt{\log n}}$) min-entropy can be done much more efficiently then from sources with high min-entropy. This is totally opposite to all previous results.

6. REFERENCES

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

[2] M. Blum. Independent unbiased coin flips from a correlated biased source: a finite state Markov chain. In *25th Annual Symposium on Foundations of Computer Science*, pages 425–433, Singer Island, Florida, 24–26 Oct. 1984. IEEE.

[3] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, Nov. 1984.

[4] J. Cai, A. Nerurkar, and D. Sivakumar. Hardness and hierarchy theorems for probabilistic quasi-polynomial time. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 726–735, 1999.

[5] B. Chor and O. Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, Mar. 1989.

[6] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag, Algorithms and Combinatorics, 1998.

[7] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 Oct. 1995. IEEE.

[8] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near optimal conversion of hardness into pseudorandomness. In *40th Annual Symposium on Foundations of Computer Science*. IEEE, 1999.

[9] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[10] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science*. IEEE, 1998.

[11] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[12] N. Nisan. Extracting randomness: How and why: A survey. In *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, pages 44–58, Philadelphia, Pennsylvania, 24–27 May 1996. IEEE Computer Society Press.

[13] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. In *Journal of Computer and System Sciencess*, 1998.

[14] N. Nisan and A. Wigderson. Hardness vs. randomness (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 2–11, White Plains, New York, 24–26 Oct. 1988. IEEE.

[15] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, Feb. 1996.

[16] J. Radhakrishnan and A. Ta-Shma. Tight bounds for depth-two superconcentrators. In *38th Annual Symposium on Foundations of Computer Science*, pages 585–594, Miami Beach, Florida, 20–22 Oct. 1997. IEEE.

[17] R. Raz, O. Reingold, and S. Vadhan. Error reduction for extractors. In *40th Annual Symposium on Foundations of Computer Science*. IEEE, 1999.

[18] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[19] M. Santha and U. V. Vazirani. Generating quasi-random sequences from slightly-random sources (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 434–440, Singer Island, Florida, 24–26 Oct. 1984. IEEE.

[20] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[21] L. Trevisan. Construction of near-optimal extractors using pseudo-random generators. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[22] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 245–251, San Diego, California, 16–18 May 1993.

[23] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 Nov. 1982. IEEE.

APPENDIX

A. PROOF OF LEMMA 1

In this section we repeat the argument of Nisan and Wigderson. We need the following theorem by Yao, which asserts that to fool general tests it is enough to fool prediction tests.

THEOREM 15. *[23] There is a polynomial time oracle Turing Machine M^A with the following property. Assume a distribution $b = (b_1, \cdots, b_m)$ on $\{0,1\}^m$ does not ϵ-fool a test A. Choose uniformly an i and an m bit string α. Then if M is run on $(b_1, \cdots, b_{1-1}; \alpha)$ using oracle A,*

$$Prob[M^A(b_1, \cdots, b_{i-1}; \alpha] \neq b_i] < 1/2 - \Omega(\epsilon/m)$$

We are now ready to construct the required Turing Machine. In our case the distribution $b_1, \cdots, b_m$ is the distribution of outputs of the generator. We use x and β to generate it as follows: We start by using x and β to construct a seed s for the generator. Let us denote the elements of S_i by $\{a_1 < \cdots < a_l\}$. For each $p \notin S_i$, we set $s_{a_p} = \beta_p$. We fill the remaining l places by setting $s_{a_p} = x_p$. We now set $b_1, \cdots, b_m$ to be the output of $NW^{f,\Delta}(s)$. (Note that s is uniformly distributed when i and x, β are uniformly distributed).

We now have the following equalities.

- $b_i = f(x)$.
- For $j < i$, $b_j = f^{\Delta}_{i,j,\beta}(s|_{S_i \cap S_j})$.

We get that for all $j < i$, b_j can be computed from x and β using $f^{\Delta}_{i,j,\beta}$. Thus, having oracle to $f^{\Delta}_{i,\beta}$ we can compute $b_1, \cdots, b_{i-1}$. We now use the Turing machine of theorem 15 to give a good estimate on $b_i = f(x)$.