Reducing the seed length in the Nisan-Wigderson generator*

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
University of Haifa
Haifa 31905, Israel
ronen@cs.haifa.ac.il

Avi Wigderson§
Department of Computer Science
Institute for Advanced Study
Einstein Drive, Princeton, NJ 08540
avi@ias.edu

April 23, 2006

Abstract

The Nisan-Wigderson pseudo-random generator [NW94] was constructed to derandomize probabilistic algorithms under the assumption that there exist explicit functions which are hard for small circuits. We give the first explicit construction of a pseudo-random generator with asymptotically optimal seed length even when given a function which is hard for relatively small circuits. Generators with optimal seed length were previously known only assuming hardness for exponential size circuits [IW97, STV01].

We also give the first explicit construction of an extractor which uses asymptotically optimal seed length for random sources of arbitrary min-entropy. Our construction is the first to use the optimal seed length for sub-polynomial entropy levels. It builds on the fundamental connection between extractors and pseudo-random generators discovered by Trevisan [Tre01], combined with the construction above.

The key is a new analysis of the NW-generator [NW94]. We show that it fails to be pseudo random only if a much harder function can be efficiently constructed from the given hard function. By repeatedly using this idea we get a new recursive generator, which may be viewed as a reduction from the general case of arbitrary hardness to the solved case of exponential hardness.

---

*This paper is based on two conference papers [ISW99, ISW00] by the same authors.

†Research Supported by NSF Award CCR-9734911, NSF Award CCR-0098197, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF Grant 97-00188

‡Part of this work was done while at the Hebrew University and the Institute for advanced study.

§This research was supported by grant number 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities and USA-Israel BSF Grant 97-00188

1 Introduction

1.1 Background

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 approaches for obtaining such a string resulted in rich and elaborate theories.

Pseudo-random generators: The first direction (introduced by [BM84, Yao82]) 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 (often referred to as *the seed*) into the appropriate length. The distribution produced should "look random" to all efficient algorithms.

Definition 1.1 (pseudo-random generators). *Let A be a predicate on m bit strings. We say that a distribution D on m bit strings* ϵ-fools *A if*

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

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*[1]*.*
- *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.*[2]

  We say that a function $G : \{0,1\}^d \to \{0,1\}^m$ is a (ϵ, m)-pseudo-random generator if the distribution $G(U_d)$ is (ϵ, m)-pseudo-random.

The reader is also referred to the excellent monograph [Gol98] devoted to this field and its varied connections to complexity theory, cryptography and learning theory. Perhaps the most important connection is that given such a generator, a simple nontrivial deterministic simulation of the given probabilistic algorithm follows. One simply runs the probabilistic algorithm on all outputs of the generator (varying over all seeds) and outputs the majority vote. As the running time of this procedure is exponential in the "seed" length it is thus crucial to reduce this parameter. Following Nisan and Wigderson we allow pseudo-random generators to run in time $2^{O(d)}$ and call such generators *explicit*[3]. We allow the running time to be exponential in the input length since in the simulation described above we intend to run the generator over all 2^d seeds anyway.

A major result in this direction was discovered in [NW94]. They showed that every difficult problem (in $E = dtime(2^{O(n)})$) 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.

[1] It is standard that a distribution D is ϵ-uniform if and only if the statistical (L1-norm) distance between D and U_m is bounded by 2ϵ.

[2] We use m both for the circuit size and the number of input bits. This is done to avoid another parameter. Note that the size of a circuit is an upper bound on the number of input bits.

[3] Naturally, we need a family of generators with varying input (and output) sizes for this to make sense.

Their work has been quantitatively improved and qualitatively extended ([BFNW93, Imp95, IW97, IW01, STV01, KvM02, CNS99]), 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 length of the given hard function, this was not achieved yet, and the best construction so far [STV01] had seeds that were nearly `quadratic`.

Extractors: The second direction (introduced by [Blu86, SV86]) asks if such algorithms can function when their random input is not independent unbiased bits, but rather the output of some "defective random source". (In practice one rarely has access to "truly random bits" and a more realistic assumption is that one can sample from a defective random source). Such a source is assumed to contain sufficient entropy, but otherwise can be arbitrary. The idea is to use procedures, called *extractors*, (defined in [NZ96]), to convert this "hidden" entropy into a (nearly) uniform distribution. It is easy to show that this task cannot be achieved by deterministic procedures, and thus extractors also get a second input: A short "seed" consisting of a small number of high quality random bits, i.e., unbiased and independent. We now give a formal definition of extractors.

Definition 1.2 (min-entropy). *A distribution P on $\{0,1\}^n$ is said to have min-entropy (at least) k, if $P(x) \leq 2^{-k}$ for every $x \in \{0,1\}^n$.*

Definition 1.3 (extractors [NZ96]). *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. We say that a family of extractors is explicit if it can be computed in polynomial time.*[4]

Given an extractor one can simulate probabilistic algorithm using *only* one sample from a high-entropy defective source. This is done by sampling an element from the source and then running the algorithm on all outputs of the extractor (varying over all 2^d seeds) and taking the majority vote. It is easy to verify that this procedure simulates the probabilistic algorithm correctly. Again, minimizing the length of the seed is crucial for the efficiency of this conversion. The reader is referred to the excellent survey papers [Nis96, NTS99] on the many varied uses of extractors in complexity, combinatorics, coding theory and network design. A more recent survey [Sha02] covers recent developments in extractor constructions.

In a breakthrough result, Trevisan ([Tre01]) showed that pseudo-random generators and extractors are intimately linked. He proved 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 of the same length under this translation. [Tre01] (and then [RRV02]) proceeded to use the NW-generator to give simple and direct constructions of extractors. For the same reasons as explained above, these constructions sometimes require nearly quadratic seed length in the "optimal" value (achieved by the non-constructive "optimal" extractor which can be proven to exist by a counting argument and has parameter matching the known lower bounds [NZ96, RTS00]). Thus, improving the NW-generator will impact both pseudo-random generators and extractors.

[4] 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.

Table 1: Pseudo-random generators comparison

Reference	seed size	circuit size
[BFNW93]	$d = O(l^4 \log^3 k)$	$m = k^{\Omega(1)}$
[Imp95]	$d = O(l^2 \log k)$	$m = k^{\Omega(1)}$
[IW97]*	$d = O(\frac{l^4}{\log^3 k})$	$m = k^{\Omega(1)}$
[STV01]	$d = O(\frac{l^2}{\log k})$	$m = k^{\Omega(1)}$
Theorem 1.4	$d = O(l)$	$m = k^{\Omega(\frac{1}{\log log(l/\log k)})}$
Ultimate goal†	$d = O(l)$	$m = k^{\Omega(1)}$

All results assume the existence of a function $f = \{f_l\}$, $f_l : \{0,1\}^l \to \{0,1\}$ which is computable in time $2^{O(l)}$, and that circuits of size $k(l)$ cannot compute.

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

†This goal has recently been achieved in [SU01, Uma02].

1.2 Results

In this paper, we construct both pseudo-random generators and extractors that have optimal seed length (by that we mean a seed length which is linear in the optimal value). Our pseudo-random generator gives an improved (and almost optimal) tradeoff between hardness and randomness. The improvement we get becomes more notable (compared to previous results) as the hardness assumption is weakened.[5]

The extractors we obtain from this construction (using Trevisan's method) have the optimal seed length *regardless* of the amount of randomness present in the imperfect random source, and are thus the first for sub-polynomial entropies to have optimal seed length. Below we state our main Theorems formalizing the above. An important note regarding both pseudo-random generators and extractors is that while we achieve optimal seed length, the output length in both cases is suboptimal. We discuss this in more detail in Section 6.

Theorem 1.4 (`Optimal seed generators`)**.** *If there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, f_l cannot be computed by circuits of size $k(l)$, then there exists functions $d(l) = O(l)$, $m(l) = k(l)^{\Omega(\frac{1}{\log\log(l/\log k)})}$ and a $(1/m(l), m(l))$-pseudo random generator $G : \{0,1\}^{d(l)} \to \{0,1\}^{m(l)}$ which is computable in time $2^{O(l)}$.*

The proof of this Theorem appears in section 4. See table 1 for comparison with previous results.

We now list our results for extractors. See table 2 for comparison with some previous results. The proofs of the next Theorems appear in section 4.

Theorem 1.5 (`Optimal seed extractors`)**.** *For every constant $\delta > 0$ and $\epsilon > exp(-n/(\log^* n)^{\log^* n})$ there is an explicitly constructible family of (k,ϵ)-extractors $Ext_{n,k,\epsilon} : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{\epsilon})} \to \{0,1\}^{k^{1-\delta} - O(\log(1/\epsilon))}$.*

We can do better for low entropy sources. We achieve a larger output in this case (whereas many previous constructions work better when the entropy is large). An example is the following

[5]We remark that our results were improved by subsequent work. See discussion in Section 6.

Table 2: Extractors comparison

reference	min-entropy k	output length m	additional randomness d
[Zuc97]	$k = \Omega(n)$	$m = (1-\delta)k$	$d = O(\log n)$
[Tre01]	any k	$m = k^{1-\delta}$	$d = O(\frac{\log^2 n}{\log k})$
Theorem 1.5	any k	$m = k^{1-\delta}$	$d = O(\log n)$
[RRV02]	any k	$m = (1-\delta)k$	$d = O(\log^2 n)$
Theorem 1.6	any k	$m = \Omega(\frac{k}{\log\log\frac{\log n}{\log k}})$	$d = O(\log n + \log^2 k)$
Corollary 1.7	$k \le 2^{\sqrt{\log n}}$	$m = (1-\delta)k$	$d = O(\log n \cdot \log\log\log n)$
Ultimate goal	any k	$m = k$	$d = O(\log n)$

The results are given for constant ϵ.

δ is an arbitrary small constant.

Theorem in which we get a larger improvement in the number of extracted bits if the entropy is small.

Theorem 1.6 (`Extractors for low min-entropy`). *For every $\epsilon > exp(-n/(\log^* n)^{\log^* n})$ there is an explicitly constructible family of (k,ϵ)-extractors $Ext_{n,k,\epsilon} : \{0,1\}^n \times \{0,1\}^{O(\log\frac{n}{\epsilon}+\log^2 k)} \to \{0,1\}^m$, for $m = \Omega(\frac{k}{\log\log(\log n/\log k)}) - O(\log(1/\epsilon))$.*

Note that when $k < 2^{\sqrt{\log n}}$ the seed length becomes the optimal $O(\log n)$. Using the method of [WZ93] the number of bits extracted can be enlarged at the expense of enlarging the seed length.[6] Using this we can improve the fraction of bits extracted to a constant fraction paying only a $O(\log\log\log n)$ penalty in the seed length.

Corollary 1.7. *For every constant $\delta > 0$ and $\epsilon > exp(-n/(\log^* n)^{\log^* n})$ there is an explicitly constructible family of (k,ϵ)-extractors $Ext_{n,k,\epsilon} : \{0,1\}^n \times \{0,1\}^{O(\log(n/\epsilon)\cdot\log\log\log n)} \to \{0,1\}^{(1-\delta)k - O(\log(1/\epsilon))}$ as long as $k < 2^{\sqrt{\log n}}$.*

1.3 A note on optimality of seed length

In [NZ96] (and later [RTS00]) it was shown that the seed length of any (k,ϵ)-extractor which outputs more bits then its seed length must be at least $\Omega(\log(n/\epsilon))$. Thus, the seed length in our extractor construction is optimal except for the exact constant.

In Section 5 we give evidence that the seed length in our construction of pseudo-random generators is also close to optimal. We now explain what we mean by this statement. We are concerned with the optimality of the *reduction* from a hard function to a pseudo-random generator. As we now explain, this poses some difficulties: Let's consider studying the case of converting a function f that is hard against say polynomial size circuits into a pseudo-random generator. Our construction

[6] The basic idea of Wigderson and Zuckerman is that if an extractor E does not extract all the randomness from the source then additional randomness can be extracted by running another extractor with an independent seed. By repeatedly running this procedure an extractor with output length $m = k/r$ and seed length d can be transformed into one with output length $m' = (1-\delta)k$ (here $\delta > 0$ is an arbitrary small constant) and seed length $d' = O(dr)$. Exact details can be found for example in [RRV02].

gives a generator which stretches l bits into $l^{O(1)}$ bits. However, it is widely believed that there are functions hard against much large circuits which in turn lead to better generators. How can we rule out the scenario in which the reduction ignores the function f altogether and uses a better available pseudo-random generators?

Below, we suggest two notions of optimality that address this difficulty and address the quality of the *reduction* from hard functions to pseudo-random generators rather than the quality of the pseudo-random generator itself.

- By Trevisan's argument ([Tre01], see also Theorem 2.8 and [KvM02]) any construction which converts hardness into a pseudo-random generator and has some additional "relativization properties" gives an extractor with the same seed length. Thus, any such construction is bounded by the lower bounds on extractors cited above. Our construction of pseudo-random generators (as well as all previous constructions) has these "relativization properties". It follows that the seed length in our pseudo-random generator construction is optimal except for the exact constant. Exact details are given in section 5.1.

- It is possible to convert pseudo-random generators into hard functions. Roughly speaking, a pseudo-random generator with seed length l which fools circuits of size $m(l)$ can be converted into a function on l bits which is hard for circuits of size $m(l)$. It follows that if a conversion of hardness into pseudo randomness (even one which does not have the "relativization properties" mentioned above) does significantly better than our construction than it also yields a harder function than the one initially supplied to it. Exact details are given in section 5.2.

1.4 Technique

This section attempts to give a complete account of the sequence of ideas used to prove our results, at a semi-intuitive, semi-technical level, that will hopefully simplify the reader's task when reading the formal proof.

Our goal is to transform a function $f : \{0,1\}^l \to \{0,1\}$ which cannot be computed by circuits of size k into a pseudo random generator with optimal seed length:

$$G : \{0,1\}^{O(l)} \to \{0,1\}^m$$

that fools circuits of size m for large as possible m. (Our final result has $m = k^{\Omega(1/\log\log(l/\log k))}$ which falls slightly short of the desired $m = k^{\Omega(1)}$). Our starting point is the NW-generator which uses a larger seed under the same assumption:[7]

$$NW : \{0,1\}^{O(\frac{l^2}{\log k})} \to \{0,1\}^m$$

The proof of Nisan and Wigderson shows how to construct a circuit C which computes the function f given any circuit A which is not fooled by their generator. The circuit C is constructed by combining

[7] This is not quite accurate. Actually the NW-generator starts from a slightly stronger assumption, that is that the function is *hard on average* for circuits of size k and not only hard on the worst case. We ignore this difference in the informal explanation because Sudan et al. [STV01] showed how to perform "hardness amplification" (that is transforming a function that is hard on the worst case into one that is hard on the average). The transformation of [STV01] doesn't significantly affect the parameters of the function and therefore in this informal discussion we always assume without loss of generality that a hard function is actually hard on average. The precise details are given in the technical part.

A with a circuit C' that computes some function f' (which we elaborate on momentarily). Such a function f' (which depends on f and A) is shown to *exist.* In order to guarantee that C is "small", Nisan and Wigderson have to make sure that f' has a small circuit C'. They achieve this by setting the parameters so that f' is a function over very few bits (and therefore has a small circuit by a trivial exponential bound). However, it turns out that the size of the input of f' *depends* on the seed length of the generator. If f is not assumed to be extremely hard, Nisan and Wigderson have to increase the seed length to guarantee that f' has a small enough circuit to contradict the hardness assumption.

The main idea of this paper is a "win-win analysis" which uses the NW-generator with the "wrong" parameters. We set the seed length to the optimal $d = O(l)$ regardless of k, that is regardless of how hard is the hard function f. (In contrast to the setting by Nisan and Wigderson in which the seed length is set to $O(l^2/\log k)$.) This is not guaranteed to produce a pseudo-random generator as the circuit C constructed by the argument of Nisan and Wigderson may be too large. However, if this happens and the Nisan-Wigderson argument fails to show that the generator is pseudo-random, then f' does not have a small circuit. More precisely, f' requires a circuit of size roughly the same as that of the best circuit for f. The parameters are chosen in a way that guarantees that the input length to f' is *smaller* than that of f (say only a half of the input size of f). Thus, (since we measure the complexity of functions relative to the length of their input), f' is actually much harder than f! We conclude that when setting the parameters this way, either we obtain a pseudo-random generator with optimal seed length or we obtain a harder function than the one initially supplied.

We then use this idea recursively trying to construct a generator from f'. Once again either we construct a pseudo random generator with optimal seed length or we obtain a harder function. Eventually, if we fail to construct a pseudo-random generator we obtain a function which is extremely hard (a function on l' bits that requires circuits of size $2^{\Omega(l')}$) and can use the standard Nisan-Wigderson analysis to construct a pseudo-random generator. Thus, our construction can be viewed as a reduction from the general case of arbitrary hardness to the solved case of exponential hardness.

The main complication when trying to implement the argument above is that the identity of the function f' is determined by a probabilistic argument – it only shows the existence of such a function but does not tell us how to find it. More precisely, the argument shows that we can index $2^{O(l)}$ functions, one of which is the required one. Considering all these potential functions at all levels of recursion, we can picture a tree. The root is labelled by the initial function f. The descendants of a node labelled by a function g are the ($2^{O(l)}$) functions which may arise as f' when using the NW-argument on the function g. We consider such a tree. By choosing the parameters carefully we get that the tree is of height $O(\log\log(l/\log k))$ and the leaves are functions on only $O(\log k)$ inputs.

One can construct a generator from all functions labelling nodes of this tree. This approach produces *many* (roughly $2^{O(l)}$) candidate generators. We claim that one of them is guaranteed to be pseudo-random. If none of the generators in the internal nodes is pseudo-random then there must be a function at a leaf which has hardness roughly k. Such a function has exponential hardness and thus the generator constructed from it is pseudo-random by the Nisan-Wigderson argument.

To complete the construction we show how to "combine" all the candidate into a single pseudo-random generator.[8] Two natural ideas come to mind.

[8]We note that at this point we can easily get a "hitting-set generator" by using an additional seed to choose a

- In every node of the above tree replace the functions $f_1, \cdots, f_{2^{O(l)}}$ by a single function with two arguments: $\bar{f}(i, x) = f_i(x)$. Certainly, if one of the candidate functions is hard so is the "concatenated" function.

- Combine all the candidate generators by running them with independently chosen seeds and take the exclusive-or of the outputs. Certainly, if one of the candidates is pseudo-random, so is the xor-generator.

Both ideas fail in our setting due to the huge number of candidates. The first idea will cause the input size of $\bar{f}$ to be significantly *larger* than that of any of the f_i's as the length of i is larger than l. We will no longer have that $\bar{f}$ is over fewer bits than f and will not be able to deduce that $\bar{f}$ is harder than its ancestor. The second idea blows the seed length by a factor of the number of candidate generators.

However, both ideas are essential components of our construction. In order to be able to use them we need to reduce the degree of the tree from $2^{O(l)}$ to poly(k), where k is the hardness of the original function f. Assume for the moment that this can be magically done. Once achieved, the first idea is applicable since the length of the added input is now at most $O(\log k)$ (which will not drastically increase the input length of functions in the tree). In other words, a tree of such a small degree (poly(k) can be collapsed into a path whose length is the depth of the tree.

Since this path is of small length $O(\log\log(l/\log k))$, we can now apply the second idea and xor these few candidate generators. We avoid even the small loss to be suffered because of their number by arranging the seed lengths of different levels to go down geometrically.

The missing important step is cutting down the degree of the tree from $2^{O(l)}$ to k^2. This requires a slightly finer inspection of the argument of Nisan and Wigderson. We show that when we fail to produce a pseudo-random generator, not only one of the descendants of the function is hard, but rather a non-negligible fraction of them, (more precisely a $1/k$-fraction). Thus, if we randomly sample k^2 descendants we almost surely "hit" a hard one if it exists. We will only run the recursion on these k^2 descendants. Of course, trivial sampling costs more random bits than we can afford, but this can be done just as well with pairwise independence. This additional randomness is added to the seed and increases the seed length of the generator only by $O(l)$ bits which we can indeed afford. This completes the argument.

There is however some price to performing this recursion. The circuit lower bound we can guarantee for f' is slightly smaller than we can guarantee for f. (If f requires circuits of size k, we can only guarantee that f' requires circuits of size $k/m^{O(1)}$). This loss is accumulated over the $\log\log(l/\log k)$ levels and causes us to have output length $m = k^{\Theta(1/\log\log(l/\log k))}$ rather than $k^{\Omega(1)}$.

This difficulty can be avoided when applying Trevisan's technique for constructing extractors. In such a case the hardness measure can be changed from circuit size to description size, (which we define formally using Kolmogorov Complexity). This is beneficial because the procedure which constructs a circuit for f given a circuit for f' can be *described* using many fewer bits than the size of the final circuit. Using this measure the hardness guarantee we get on f' is much larger, ($k - O(m)$ instead of $k/m^{O(1)}$), and as a result, we can increase m and extract any polynomial fraction of the initial entropy and at the same time simplify the argument.

---

random candidate.

1.5 History of this paper

This paper is the combination of two conference papers by the same authors. The first paper, [ISW99] included the idea of using the NW-generator with the wrong parameters and performing the "win-win" analysis. We then constructed all candidate generators in the tree, and showed how to combine them into a hitting set generator with optimal seed length. This in turn suffices for derandomizing two-sided error probabilistic algorithms using the results of [ACR96, ACRT99]. We also gave a more direct argument in which we conduct a "tournament" between all the candidate pseudo-random generators. While we cannot guarantee that the winning candidate is a pseudo-random generator, we can guarantee that it could be used to derandomize a *given* probabilistic algorithm on a *given* input. This gives a derandomization of BPP.

The second paper, [ISW00] performed the combination of candidate generators into a single pseudo-random generator and constructed optimal seed extractors. The version presented here is mostly based on that paper.

1.6 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. In section 5 we discuss notions of optimality for pseudo-random generators. In section 6, we discuss open problems and mention subsequent work.

2 Definitions and Ingredients

Notation 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 $G(U_d)$ to denote the distribution obtained by applying G on a random sample from U_d.

2.1 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 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. This is why we define circuit complexity and Kolmogorov complexity relative to a predicate A.

Definition 2.1 (circuit complexity). *Let A be a predicate on m bit inputs.*

worst case: *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). We use $S(f)$ to denote the circuit complexity of f.*

average case: *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 *the average* over a random input. 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 2.2 (Kolmogorov Complexity). *Let A be a predicate on m bit inputs.*

worst case: *Define $K_A(f)$, the Kolmogorov complexity of f given A, as the length (in bits) of the smallest description of an oracle Turing Machine which, using oracle A, outputs f.*

average case: *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.*

2.2 Pseudo-random generator schemes

Trevisan, [Tre01] discovered that any conversion of hard functions into pseudo-random generators with certain "relativization properties" also gives extractors. The following definition summarizes these properties.

Definition 2.3. *A (k,ϵ)-pseudo-random generator scheme is a function $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property: For any $f \in \{0,1\}^n$ and predicate A on m bits, if G^f does not ϵ-fool A then $S_A(f) \leq k$. We say that a family of pseudo-random generator schemes is explicit if it can be computed in time polynomial in n.*

Note that given an explicit pseudo-random generator scheme any hard function can be used to construct a pseudo-random generator by "plugging" the truth table of the hard function as the first input to the pseudo-random generator scheme. We use the following notation:

Definition 2.4. *For a function $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ and $h \in \{0,1\}^n$ we define $G^h : \{0,1\}^d \to \{0,1\}^m$ by $G^h(x) = G(h,x)$ and call it* the generator derived from G by h.

The following Lemma immediately follows.

Lemma 2.5. *Assume there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, $S(f_l) > k(l)$. Furthermore assume there is an explicit $(k(l)/m(l), \epsilon(l))$-pseudo-random generator scheme $G : \{0,1\}^{2^l} \times \{0,1\}^{d(l)} \to \{0,1\}^{m(l)}$ with $d(l) \geq l$. Then $G^f : \{0,1\}^{d(l)} \to \{0,1\}^{m(l)}$ is a $(\epsilon(l), m(l))$-pseudo-random generator which is computable in time $2^{O(l)}$.*

We remark that the condition that $d(l) \geq l$ is unnecessary as we later show that any pseud-random generator scheme has $d(l) \geq l$.

Proof. (of Lemma 2.5). For every l we consider $G'_l = G^{f_l}$. That is we supply the truth table of f_l as the first input of G. We compute G'_l by first computing the truth table of f_l (which takes time $2^{O(l)}$) and then feeding it to G. The total time is polynomial in 2^l. If $G'_l(\cdot)$ isn't (ϵ, m)-pseudo-random then there's a circuit A of size m which is not ϵ-fooled by G^{f_l}. It follows that $S_A(f_l) \leq k(l)/m(l)$ which gives $S(f_l) \leq k(l)$. □

Trevisan observed that all known hardness versus randomness tradeoffs actually construct explicit pseudo-random generator schemes. Our construction is no exception. He also proved that any such scheme is an explicit extractor. It turns out that for this connection it is more natural to replace circuit complexity by Kolmogorov complexity.

Definition 2.6. *A (k, ϵ)-extractor scheme is a function $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property: For any $f \in \{0,1\}^n$ and predicate A on m bits, if G^f does not ϵ-fool A then $K_A(f) \le k$.*

It immediately follows that a pseudo-random generator scheme is an extractor scheme.

Lemma 2.7. *Any (k, ϵ)-pseudo-random generator scheme is a $(O(k \log k), \epsilon)$-extractor scheme.*

Proof. This follows because any circuit of size k can be described by $O(k \log k)$ bits. □

The converse is not necessarily true. The following Theorem, (by [Tre01]) asserts that extractor schemes are extractors.

Theorem 2.8. *(implicit in [Tre01]) Any (k, ϵ)-extractor scheme is a $(k + \log(1/\epsilon), 2\epsilon)$-extractor.*

Proof. Let Ext be a (k, ϵ)-extractor scheme, 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 A on Ext^h. 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 ϵ. Therefore, the total bias is at most 2ϵ. □

There is also a partial converse:

Lemma 2.9. *Any explicit (k, ϵ)-extractor is a $(k + O(1), \epsilon)$-extractor scheme.*

Proof. Let Ext be a (k, ϵ)-extractor. 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 $K_A(h) \le \log |H_A| + O(1)$ 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 $\log |H_A| < k + 1$. □

Following Trevisan, we use Theorem 2.8 to reduce the problem of constructing extractors into proving hardness versus randomness tradeoffs.

2.3 The Nisan-Wigderson generator

Almost all schemes of converting hardness into pseudo-randomness, as well as some extractor constructions, use the NW-generator from [NW94]. 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.

Definition 2.10 (designs). *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 2.11 (The NW-generator). *Let $l < d < m$ be integers, and let $n = 2^l$. Let Δ be a (l, u)-design. Define a function $NW^\Delta : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ 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 $\{0,1\}^d$ to $\{0,1\}^m$ 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 in (n, d) and in particular $NW^{f,\Delta}(x)$ can be computed, given Δ, in time polynomial in m, with an oracle for f.

Theorem 2.12. *(see for example, Lemma 13 in [RRV02]) 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 $poly(2^d)$.*

Remark 2.13. *It can be shown that the designs constructed by Nisan and Wigderson are optimal up to constants for the parameters we are interested in. Thus, it is impossible to improve the seed length of the NW-generator by constructing more efficient designs. More precisely, shooting for the optimal seed length $d = O(l)$, it has to be the case that $u = \Omega(l)$. This follows directly from the inclusion-exclusion formula, see [ISW99] for details.*

In [RRV02] it was observed that a weaker combinatorial property (which they call "weak designs") is sufficient for performing the analysis of Nisan and Wigderson. Weak designs are better than designs when u is very small. However, the lower bound of $u = \Omega(l)$ when $d = O(l)$ applies also to weak designs. Our construction can be carried out with weak designs but this does not improve the parameters of our pseudo-random generators/extractors.

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

Definition 2.14. *Given an (l, u)-design $\Delta = (S_1, \cdots, S_m \subseteq [d])$ and a function $f : \{0,1\}^l \to \{0,1\}$, 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:*

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

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 $f^{\Delta}_{i,\beta}$ can be computed in time $poly(2^l)$. These functions are over $u + \log m$ bits, which trivially entails that $S(f^{\Delta}_{i,\beta})$ and $K(f^{\Delta}_{i,\beta})$ are bounded by $m2^u$.

The following Lemma is implicit in [NW94][9]:

Lemma 2.15. *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*

$$|Prob[M^{g,A}(x, i, \alpha, \beta) = f(x)] - 1/2| = \Omega(\epsilon/m)$$

The conclusion of Lemma 2.15 is that when given oracle access to a test A which is not fooled by the NW-generator we can use g to compute f (or the negation of f) correctly on a $1/2 + \Omega(\epsilon/m)$ fraction of the inputs. For completeness we give a proof of Lemma 2.15. We need the following theorem which is a variant of the "distinguisher to next-bit predictor" lemma of [Yao82, GM84].[10]

Theorem 2.16. *[Yao82] 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 $(b_1, \cdots, b_{i-1}; \alpha)$ using oracle A,*

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

Proof. (of Lemma 2.15) Let $(b_1, \cdots, b_m)$ denote the distribution $NW^{f,\Delta}$ and let A be a test that is not ϵ-fooled by this distribution. We now explain how to construct the machine M. Recall that the machine is given uniformly chosen (x, i, α, β). We first think of i and α as fixed and show how to use the oracle access to $f^{\Delta}_{i,\beta}$ to generate the distribution $b_1, \ldots, b_i$ from the randomly chosen x and β. 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 $s_p = \beta_p$. We fill the remaining l places with x by setting $s_{a_v} = x_v$. Note that s is uniformly distributed when x, β are uniformly distributed and therefore the distribution $NW^{f,\Delta}(s)$ is identical to $(b_1, \cdots, b_m)$. We now have the following equalities.

[9] [NW94] 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 original argument of [NW94] can be presented this way: [NW94] uses designs with very small u, (which in turn forces d to be relatively large). This makes the circuit complexity of all the $f^{\Delta}_{i,\beta}$'s relatively small. Lemma 2.15 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 $Size_m$.

The observation that the proof of Nisan and Wigderson relativizes was made in [KvM02]. This observation is important for Trevisan's extractor, [Tre01]. The observation added here is that the argument of Nisan and Wigderson connects the complexity of f and its restrictions.

[10] The standard variants of this argument transforms a distinguisher for the distribution $b_1, \cdots, b_m$ into a predictor which when given $b_1, \cdots, b_{i-1}$ predicts the next bit b_i. In this variant we only demand that the predictor output is correlated with the bit b_i (but not necessarily positively correlated). Nevertheless, the standard proof of this theorem works by first achieving this goal and then choosing whether to keep the output bit or flip it. We settle for this weaker notion because we want a uniform reduction which uses few random bits. However, we could have replaced the machines in both Theorem 2.16 and Lemma 2.15 with nonuniform circuits which would have allowed us to use the standard version and get almost the same results.

- $b_i = f(x)$.
- 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 2.16 to give a good estimate on $b_i = f(x)$. □

The Lemma above can be used to connect the complexity of the original function, that of the restricted functions, and the power of the generator.

Corollary 2.17. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i:* $S_{A,1/2-\Omega(\epsilon/m)}(f) \le S_A(f^{\Delta}_{i,\beta}) \cdot m^{O(1)}$

Corollary 2.17 gives that if $NW^{f,\Delta}$ does not fool A, then some 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). As we explained in the introduction, this is useful since all the restrictions are on only $u + \log m$ bits.

Proof. (of corollary 2.17) By Lemma 2.15 if $NW^{f,\Delta}$ does not ϵ-fool A, then by averaging there exists an $\alpha \in \{0,1\}^m$ such that:

$$|Prob_{x,i,\beta}[M^{f^{\Delta}_{i,\beta},A}(x,\alpha,\beta) = f(x)] - 1/2| = \Omega(\epsilon/m)$$

By a Markov argument this gives that for a $\Omega(\epsilon/m)$ fraction of pairs (β, i):

$$|Prob_x[M^{f^{\Delta}_{i,\beta},A}(x,\alpha,\beta) = f(x)] - 1/2| = \Omega(\epsilon/m)$$

Fix such a pair of β and i. We have that the output on a uniformly chosen x is correlated with $f(x)$. The equation above can be expressed in the following form: There is a bit $b \in \{0,1\}$ so that

$$Prob_x[M^{f^{\Delta}_{i,\beta},A}(x,\alpha,\beta) \oplus b = f(x)] \ge 1/2 + \Omega(\epsilon/m)$$

By "hard-wiring" α, b, β, i and the circuit for $f^{\Delta}_{i,\beta}$ to M, we get the circuit promised in the Corollary. □

An analogous argument can be used to state Corollary 2.17 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. We only have to specify (α, b, β, i). The total length of these strings is $m + O(\log m) \le 2m$. Note that as we only need to specify these strings we get an additive term and not a multiplicative term.

Corollary 2.18. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i:* $K_{A,1/2-\Omega(\epsilon/m)}(f) \le K_A(f^{\Delta}_{i,\beta}) + 2m$

2.4 Xoring generators

We need a technique for combining many candidate generators where one of them is guaranteed to fool a test A. We'd like to "xor" them all on independent seeds to get 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 2.19. *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 2.20. *Let $P_1, \cdots, P_r$ be distributions on $\{0,1\}^m$ and $A : \{0,1\}^m \to \{0,1\}$ 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.*

Proof. Suppose P_i is ϵ-pseudo-random for $A^{\oplus}$. Without loss of generality, assume $i = r$.

$$Pr_{z \in_{\bar{P}} \{0,1\}^m}(A(z) = 1) = E_{z_1 \in_{P_1} \{0,1\}^m, \cdots, z_{r-1} \in_{P_{r-1}} \{0,1\}^m}, \left[Pr_{z_r \in_{P_r} \{0,1\}^m}(A(z_1 \oplus \cdots \oplus z_r) = 1)\right]$$

Fix $z_1, \cdots, z_{r-1}$ arbitrarily and let $y = z_1 \oplus \cdots \oplus z_{r-1}$. We know that P_r ϵ-fools $A \oplus y$, and therefore

$$|Pr_{z_r \in_{P_r} \{0,1\}^m}(A(z_1 \oplus \cdots \oplus z_r) = 1) - Pr_{z \in_R \{0,1\}^m}(A(z) = 1)| \leq \epsilon$$

Taking expectation over $z_1, \cdots, z_{r-1}$, we get that $\bar{P}$ ϵ-fools A. □

The price of Lemma 2.20 is that if you have two candidate generators $G_1 : \{0,1\}^{d_1} \to \{0,1\}^m$, $G_2 : \{0,1\}^{d_2} \to \{0,1\}^m$ 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.

Let us rephrase corollaries 2.17, 2.18, 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 2.21. *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 2.22. *If $NW^{f,\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i: $K_{A,1/2-\Omega(\epsilon/m)}(f) \leq K_A(f^{\Delta}_{i,\beta}) + 4m$*

2.5 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 [NW94] concerns constructing a hard to approximate function from one that is hard to compute in the worst-case ([BFNW93, Imp95, IW97, STV01]). This process is usually called hardness amplification. Here, we use the hardness amplification from [STV01], which is nearly optimal.

Theorem 2.23. *[STV01] There exists a polynomial time algorithm that given a function $f : \{0,1\}^l \to \{0,1\}$ (encoded as a 2^l bit truth table) and $\rho > 2^{-l}$, produces the truth table of a function $\hat{f} : \{0,1\}^{4l} \to \{0,1\}$, with the following properties for any predicate A:*

1. $S_A(f) \leq S_{A,1/2-\rho}(\hat{f})(\frac{l}{\rho})^{O(1)}$
2. $K_A(f) \leq K_{A,1/2-\rho}(\hat{f}) + O(\log \frac{1}{\rho})$

Remark 2.24 (List decoding and Kolmogorov Complexity). *The first item in Theorem 2.23 explicitly appears in [STV01]. The second item which concerns Kolmogorov complexity immediately follows from the construction of [STV01] as we now explain.*

A list decodable code is a mapping C from n bits to $\bar{n} > n$ bits such that every $A \in \{0,1\}^{\bar{n}}$ has few $f \in \{0,1\}^n$ such that $C(f)$ is close to A in Hamming distance. In other words given a list decodable code and $A \in \{0,1\}^{\bar{n}}$ any f for which $\hat{f} = C(f)$ is close to A has small description size $K_A(f)$. The construction of [STV01] is of a list decodable code C which runs in polynomial time. The second item in Theorem 2.23 follows.

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.

Corollary 2.25. *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: $S_A(f) \leq S_A(\hat{f}^{\Delta}_{i,\beta})(m/\epsilon)^{O(1)}$*

Corollary 2.26. *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: $K_A(f) \leq K_A(\hat{f}^{\Delta}_{i,\beta}) + 4m + O(\log \frac{m}{\epsilon})$*

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 A recursive application of the Nisan-Wigderson generator

In this Section we give our main construction and prove its correctness.

3.1 The construction

We use the same construction both for pseudo-random generators and extractors. Our goal is to construct a function $Rec : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$. Below we explain how to construct the function Rec when given parameters n and m and a function $Base$ that serves as the base of the recursion.

Ingredients:

- An integer n (the desired length of the first input of Rec).
- An integer $m < n$ (the desired output length of Rec).
- A function $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$. This is going to be an extractor/pseudo-random generator that is going to be the base of the recursion.

Result: A function $Rec : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ where $d = O(\log n) + d_{Base}$.

The inputs of Rec**:** The function $Rec : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ gets inputs $f \in \{0,1\}^n$ and $y \in \{0,1\}^d$. Let $l = \log n$. Let c_3 be a constant to be determined later, and let r be a parameter to be determined later. We define a sequence of integers $d_1, \cdots, d_r$ by $d_t = c_3 l/2^{t-1} = c_3 \log n/2^{t-1}$. We interpret the string y as a sequence of three parts defined as follows:

- A string s_{Base} of length d_{Base}.
- A string s composed of r strings $s_1, \cdots, s_r$ where for each $1 \le t \le r$, $s_t \in \{0,1\}^{d_t}$. Note that as the d_i's are a geometric progression the total length of s is bounded by $2d_1 = 2c_3 \log n$.
- A string p composed of r strings $p_1, \cdots, p_r$ where for each $1 \le t \le r$, $p_t \in \{0,1\}^{4d_t}$. Note that the total length of p is bounded by $8d_1 = 8c_3 \log n$.

By this choice $d = 10c_3 \log n + d_{Base}$ which is $O(\log n) + d_{Base}$ as we have promised.

The operation of Rec**:** The function Rec constructs $r+1$ function $f_1, \cdots, f_{r+1}$. This is done recursively as follows:

- Set $f_1 = f$ and $l_1 = l$.
- For $t > 1$ we now explain how to construct $f_{t+1} : \{0,1\}^{l_{t+1}} \to \{0,1\}$ from $f_t : \{0,1\}^{l_t} \to \{0,1\}$.
- Let $\hat{l_t} = 4l_t$. Apply Theorem 2.23 on f_t and obtains the function $\hat{f_t} : \{0,1\}^{\hat{l_t}} \to \{0,1\}$.
- Let $u_t = max(\frac{c_1 \hat{l_t}^2}{d_t}, \log m)$ where c_1 is the constant from Theorem 2.12. Apply Theorem 2.12 to construct a $(\hat{l_t}, u_t)$-design Δ_t using d_t as the "d-parameter" of the design (recall that this measures the size of the universe in which the sets live).
- We define $z_t = NW^{\hat{f_t}, \Delta_t}(s_t)$ (that is applying the Nisan-Wigderson generator with the (hardness amplified) version of f_t using the seed s_t).
- Use p_t as randomness to pick m^4 elements in $[m] \times \{0,1\}^{d_t}$ in a pairwise independent way. Let's denote these elements by $(i_{t,q}, \beta_{t,q})$ for $1 \le q \le m^4$.[11]
- For each $1 \le q \le m^4$ we consider the function $(\hat{f_t})^{\Delta_t}_{(i_{t,q},\beta_{t,q})} : \{0,1\}^{\log m + u_t} \to \{0,1\}$ defined in Definition 2.14. We then define $f_{t+1} : \{0,1\}^{4\log m + \log m + u_t} \to \{0,1\}$ by:

$$f_{t+1}(q,j,z) = (\hat{f_t})^{\Delta_t}_{(i_{t,q},\beta_{t,q})}(j,z)$$

  We define $l_{t+1} = 5\log m + u_t$ (so that l_{t+1} is the input length of f_{t+1}).

[11] Note that using linear functions (cf. [CG89]) choosing less than 2^a pairwise independent elements in $\{0,1\}^a$ can be done using $2a$ bits. Thus, we need $2(d_t + \log m)$ random bits. Recall that $d_t \ge u_t \ge \log m$ and therefore the total number of random bits required is at most $4d_t$ and we can use p_t for this purpose.

- We showed how to define l_{t+1}, u_{t+1} and f_{t+1} using l_t, u_t and f_t. We continue this process recursively and let r be the first integer so that $u_r = \log m$. The recursion stops at this point having defined f_{r+1}.

Finally, let $z_{Base} = Base^{f_{r+1}}(s_{Base})$ and the final output of Rec is $z_1 \oplus z_2 \oplus ... \oplus z_r \oplus z_{Base}$.

As explained in the introduction, the main idea in the construction above is to maintain that if the Nisan-Wigderson generator fails to fool a test A when applied with the function $\hat{f}_t$ then assuming f_t is "hard to compute/describe using A" we get that f_{t+1} is even harder.

Before showing that the construction produces pseudo-random generators and extractors we first need to verify that the construction above is well defined. That is that the recursive process above terminates. This is done in the next lemma that gives a bound on r (the number or applications of the recursion). In addition the lemma also asserts that when the recursion stops the last function f_r is over only $O(\log m)$ bits.

Lemma 3.1. *The constant c_3 in the construction can be chosen so that for any n, m the recursion terminates, and when it does*

$$r \leq \log \log \frac{\log n}{\log m}$$

and $l_{r+1} = 6 \log m$.

Proof. Recall that the recursion continues as long as $u_t > \log m$. We now study the behavior of u_t in these steps. We first derive a bound on u_{t+1} as a function of u_t. Recall that $l_{t+1} = u_t + 5 \log m \leq 6u_t$ assuming the recursion does not stop at this step. It follows that $\hat{l}_{t+1} = 4l_t \leq 24u_t$ and u_{t+1} is given by $\frac{c_1 \hat{l}_{t+1}^2}{d_{t+1}} \leq \frac{c_1 (24u_t)^2 \cdot 2^t}{c_3 l}$ where here we also used the fact that $d_t = c_3 l / 2^{t-1}$.

Suppose we have for some t that

$$u_t \leq \frac{c_1^{i_t} \cdot (24l)^{j_t} \cdot 2^{k_t}}{(c_3 l)^{i_t}} \tag{1}$$

(Note that this is indeed the case for $t = 1$ with $i_t = 1$, $j_t = 2$ and $k_t = 0$). Then using the bound we derived for u_{t+1} we get that:

$$u_{t+1} \leq \frac{c_1^{2i_t+1} \cdot (24l)^{2j_t} \cdot 2^{2k_t+t}}{(c_3 l)^{2i_t+1}}$$

We conclude that (1) holds for all $t \leq r$ with:

- $i_1 = 1$ and $i_{t+1} = 2i_t + 1$ which gives $i_t = 2^t - 1$.
- $j_1 = 2$ and $j_{t+1} = 2j_t$ which gives $j_t = 2^t$
- $k_1 = 0$ and $k_{t+1} = 2k_t + t$ which gives

$$k_t = \sum_{1 \leq i \leq t-1} 2^{i-1}(t-i) \leq 2^{t-1} \sum_{1 \leq i \leq t-1} 2^{-(t-i)}(t-i) \leq 2^{t-1} \sum_{1j \leq j \leq t-1} 2^{-j} j \leq c_4 2^t$$

For some constant $c_4 > 0$.

Thus, we have that:

$$u_t \leq \frac{c_1^{2^t-1} \cdot (24l)^{2^t} \cdot 2^{c_4 2^t}}{(c_3 l)^{2^t-1}} \leq 24 \cdot 2^{c_4} \cdot l \cdot \left(\frac{c_1 \cdot 2^{c_4}}{c_3}\right)^{2^t-1}$$

We can pick c_3 large enough as a function of c_1, c_4 so that

$$u_t \leq \frac{l}{2^{2^t}} \tag{2}$$

It follows that there is an $r \leq \log\log \frac{l}{\log m} = \log\log \frac{\log n}{\log m}$ such that $u_r = \log m$ and the recursion stops. By definition $l_{r+1} = u_r + 5\log m = 6\log m$. $\square$

The Lemma above also implies that *Rec* can be computed in time polynomial in n.

Remark 3.2. *It is instructive to consider what would have happened if we had set up the parameters in a different way. The more obvious choice would have been to use the "same design parameters" in all applications. That is choose $d_t = c_3 l_t$ for all t and then use the design with these parameters. This choice is easier to calculate. We get that $l_{t+1} = l_t/2$ which gives $u_t = l/2^t$. This would make the recursion run for more steps (as with our settings we get that $u_t = l/2^{2^t}$). Intuitively, we want to keep d_t fixed to $c_3 l$ during the recursion and benefit from the fact that l_t decreases quickly. Nevertheless, we also want to have the property that $\sum d_t = O(l)$. To achieve this we arrange d_t as a geometric series. The point is that d_t decreases much slower than l_t and therefore we get the same effect as if d_t was fixed.*

3.2 Correctness of the construction

We now state two lemmas showing that our construction yields pseudo-random generators and extractors when applied with the correct ingredients.

Lemma 3.3 (Constructing pseudo-random generator schemes). *There exists some constant $c > 0$ such that for every integers n, m and k_{Base}, if Base is an explicit $(k_{Base}, 1/2m)$-pseudo-random generator scheme $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$, then Rec is an explicit $(k_{Base} m^{cr}, 1/m)$-pseudo random generator scheme with $r \leq \log\log \frac{\log n}{\log m}$.*

Lemma 3.4 (constructing extractor schemes). *There exists some constant $c > 0$ such that for every integers n, m and k_{Base}, if Base is an explicit $(k_{Base}, 1/2m)$-extractor $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$, then Rec is an explicit $(k_{Base} + crm, 1/m)$-extractor scheme $Rec : \{0,1\}^n \times \{0,1\}^{O(\log n) + d_{Base}} \to \{0,1\}^m$ with $r \leq \log\log \frac{\log n}{\log m}$.*

By Theorem 2.8 any extractor scheme is an extractor and so we can rephrase Lemma 3.4 in the following form:

Lemma 3.5 (constructing extractors). *There exists some constant c such that for every integers n, m and k_{Base}, if Base is an explicit $(k_{Base}, 1/2m)$-extractor $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$, then Rec is an explicit $(k_{Base} + crm, 2/m)$-extractor $Rec : \{0,1\}^n \times \{0,1\}^{O(\log n) + d_{Base}} \to \{0,1\}^m$ with $r \leq \log\log \frac{\log n}{\log m}$.*

In both lemmata above, when shooting for output length m one needs to use a function *Base* with output length m (which forces $k_{Base} \geq m$). The final quality of *Rec* (that is its k parameter) is larger than k_{Base} by a factor that depends on the number of levels of the recursion.

3.3 Proof of correctness for pseudo-random generator schemes

We now give the proof of correctness for pseudo-random generator schemes.

Proof. (of Lemma 3.3) Let r be the number of steps in the recursion. By Lemma 3.1 we have that $r \leq \log\log\frac{\log n}{\log m}$. We need to show that there exists a constant $c > 0$ such that Rec is a $(k_{Base}m^{cr}, 1/m)$-pseudo-random generator scheme. Let c be a constant to be chosen later and let $k = k_{Base}m^{cr}$. It remains to show that given any $f \in \{0,1\}^n$ and a predicate A on m bits such that $S_A(f) > k$ then Rec^f $1/m$-fools A.

For every $1 \leq t \leq r$ we define the following two events over the probability space of choosing $p = (p_1, \ldots, p_r)$:

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

Claim 1. *For every* $1 \leq t \leq r$, $\Pr_{p=(p_1,\ldots,p_r)}[E_t^1 \cup E_t^2] \geq 1 - O(1/m^2)$.

Proof. (of claim 1) Fix some $1 \leq t \leq r$. The function f_t is a random variable that depends only on $p_1, \ldots, p_{t-1}$. It is sufficient to show that for any fixed values $p_1, \ldots, p_{t-1}$, $\Pr_{p_t}[E_t^1 \cup E_t^2] \geq 1 - O(1/m^2)$ (where the difference is that the probability is only over the choice of p_t). By Corollary 2.25 there exists a constant c (hidden in the $O(\cdot)$ notation of the corollary) such that if E_t^1 does not occur then for a $\Omega(1/m^2)$ of the pairs $\beta \in \{0,1\}^{d_t}$ and $i \in [m]$ we have that

$$S_A(f_t) \leq S_A((\hat{f}_t)_{i,\beta}^{\Delta_t})m^c \tag{3}$$

Recall that in the definition of f_{t+1} we use p_t to sample m^4 pairs $(i_{t,1}, \beta_{t,1}), \cdots, (i_{t,m^4}, \beta_{t,m^4})$ in a pairwise independent way. For $1 \leq q \leq m^4$ let R_q denote the indicator random variable for the event that $(i_{t,q}, \beta_{t,q})$ fulfil the condition in (3) and let $R = \sum_q R_q$ denote the number of good pairs. Note that the expected number of good pairs is $\Omega(m^2)$ and that the R_q's are pairwise independent. Chebichev's inequality gives that the probability that a sum of pairwise independent indicator random variables is zero is bounded from above by $1/\mu$ where μ is the expectation of the sum. We conclude that with probability $1 - O(1/m^2)$ there exists a q such that

$$S_A(f_t) \leq S_A((\hat{f}_t)_{i_{t,q},\beta_{t,q}}^{\Delta_t})m^c$$

Note that as f_{t+1} "contains" $(\hat{f}_t)_{i_{t,q},\beta_{t,q}}^{\Delta_t}$ we have that given a circuit C (with A gates) that computes f_{t+1} we can construct a circuit C' (with A gates) of the same size that computes $(\hat{f}_t)_{i_{t,q},\beta_{t,q}}^{\Delta_t}$. Thus, we get that:

$$S_A(f_t) \leq S_A(f_{t+1})m^c$$

and E_t^2 occurs as required. □

We define $E^1 = \cup_{1 \leq t \leq r} E_t^1$ and $E^2 = \cap_{1 \leq t \leq r} E_t^2$. By the Claim above we have that $\Pr_{p=(p_1,\cdots,p_r)}[E^1 \cup E^2] \geq 1 - \Omega(r/m^2) > 1 - 1/2m$. Consider some fixed $p = (p_1, \cdots, p_r)$. It is sufficient to show that for a $(1 - 1/2m)$-fraction of such values $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A as this means that Rec^f $1/m$-fools A as required.

If a fixing $p = (p_1, \cdots, p_r)$ makes E^1 occur then there is a $1 \leq t \leq r$ such that $NW^{\hat{f}_t, \Delta_t}$ is $1/2m$-pseudo-random for $A^{\oplus}$. For a fixed value of $p = (p_1, \cdots, p_r)$ the function Rec outputs the exclusive-or of $r+1$ independent distributions $z_1, \cdots, z_r$ and z_{Base} (each generated using its own seed). We have that z_t is distributed like $NW^{\hat{f}_t, \Delta_t}$ (when varying over the choice of s_t) and therefore by Lemma 2.20 we conclude that $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A.

We now consider the case that the fixing $p = (p_1, \cdots, p_r)$ makes E^2 occur. If this is the case then

$$S_A(f_1) \leq S_A(f_{r+1}) m^{c \cdot r}$$

It follows that $S_A(f_{r+1}) \geq k/m^{c \cdot r} \geq k_{Base}$ and thus, by the assumption that $Base$ is a $(k_{base}, 1/2m)$-pseudo-random generator scheme we have that $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A. □

3.4 Proof of correctness for extractor schemes

We now give the proof of correctness for extractor schemes. The argument is similar to that of pseudo-random generator schemes with the exception that we measure hardness by Kolmogorov Complexity. We give the full proof for completeness.

Proof. (of Lemma 3.3) Let r be the number of steps in the recursion. By Lemma 3.1 we have that $r \leq \log\log \frac{\log n}{\log m}$. We need to show that there exists a constant $c > 0$ such that Rec is a $(k_{Base} + rcm, 1/m)$-extractor scheme. We choose $c = 5$ and let $k = k_{Base} + rcm$. It remains to show that given any $f \in \{0,1\}^n$ and a predicate A on m bits such that $K_A(f) > k$ then Rec^f $1/m$-fools A.

For every $1 \leq t \leq r$ we define the following two events (over the probability space of choosing $p = (p_1, \ldots, p_r)$:

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

Claim 2. *For every* $1 \leq t \leq r$, $\Pr_{p=(p_1,\ldots,p_r)}[E_t^1 \cup E_t^2] \geq 1 - O(1/m^2)$.

Proof. (of claim 2) Fix some $1 \leq t \leq r$. The function f_t is a random variable that depends only on $p_1, \ldots, p_{t-1}$. It is sufficient to show that for any fixed values $p_1, \ldots, p_{t-1}$, $\Pr_{p_t}[E_t^1 \cup E_t^2] \geq 1 - O(1/m^2)$ (where the difference is that the probability is only over the choice of p_t). By Corollary 2.26 if E_t^1 does not occur then for a $\Omega(1/m^2)$ of the pairs $\beta \in \{0,1\}^{d_t}$ and $i \in [m]$ we have that

$$K_A(f_t) \leq K_A((\hat{f}_t)_{i,\beta}^{\Delta_t}) + cm \tag{4}$$

Recall that in the definition of f_{t+1} we use p_t to sample m^4 pairs $(i_{t,1}, \beta_{t,1}), \cdots, (i_{t,m^4}, \beta_{t,m^4})$ in a pairwise independent way. For $1 \leq q \leq m^4$ let R_q denote the indicator random variable for the event that $(i_{t,q}, \beta_{t,q})$ fulfil the condition in (4) and let $R = \sum_q R_q$ denote the number of good pairs. Note that the expected number of good pairs is $\Omega(m^2)$ and that the R_q's are pairwise independent. Chebichev's inequality gives that the probability that a sum of pairwise independent indicator random variables is zero is bounded from above by $1/\mu$ where μ is the expectation of the sum. We conclude that with probability $1 - O(1/m^2)$ there exists a q such that

$$K_A(f_t) \leq K_A((\hat{f}_t)_{i_{t,q},\beta_{t,q}}^{\Delta_t}) + cm$$

Note that as f_{t+1} "contains" $(\hat{f}_t)^{\Delta_t}_{i_{t,q},\beta_{t,q}}$ we have that given a machine with oracle access to A that describes f_{t+1} we can construct a machine with oracle access to A that computes $(\hat{f}_t)^{\Delta_t}_{i_{t,q},\beta_{t,q}}$. Thus, we get that:

$$K_A(f_t) \leq K_A(f_{t+1}) + cm$$

and E^2_t occurs as required. $\square$

We define $E^1 = \cup_{1 \leq t \leq r} E^1_t$ and $E^2 = \cap_{1 \leq t \leq r} E^2_t$. By the Claim above we have that $\Pr_{p=(p_1,\cdots,p_r)}[E^1 \cup E^2] \geq 1 - \Omega(r/m^2) > 1 - 1/2m$. Consider some fixed $p = (p_1, \cdots, p_r)$. It is sufficient to show that for a $(1 - 1/2m)$-fraction of such values $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A as this means that Rec^f $1/m$-fools A as required.

If a fixing $p = (p_1, \cdots, p_r)$ makes E^1 occur then there is a $1 \leq t \leq r$ such that $NW^{\hat{f}_t,\Delta_t}$ is $1/2m$-pseudo-random for $A^{\oplus}$. For a fixed value of $p = (p_1, \cdots, p_r)$ the function Rec outputs the exclusive-or of $r+1$ independent distributions $z_1, \cdots, z_r$ and z_{Base} (each generated using its own seed). We have that z_t is distributed like $NW^{\hat{f}_t,\Delta_t}$ (when varying over the choice of s_t) and therefore by Lemma 2.20 we conclude that $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A.

We now consider the case that the fixing $p = (p_1, \cdots, p_r)$ makes E^2 occur. If this is the case then

$$K_A(f_1) \leq K_A(f_{r+1}) + mcr$$

It follows that $K_A(f_{r+1}) \geq k - mcr \geq k_{Base}$ and thus, by the assumption that $Base$ is a $(k_{base}, 1/2m)$-extractor and using Lemma 2.9 we get that $Rec^f(U_{d_{Base}}, U_{d_1+\cdots+d_r}, p_1, \cdots, p_r)$ $1/2m$-fools A. $\square$

4 Proof of the main Theorems

In this section we show how "plugging" different bases to our construction gives the previously stated Theorems.

4.1 Pseudo-random generators

The generator of Theorem 1.4 can be achieved by using the generator of [IW97] (or the simplification of [STV01]) as $Base$. The generator of [STV01] works by first applying their hardness amplification scheme (Theorem 2.23) on the hard function f to obtain a function $\hat{f}$ and then using the Nisan-Wigderson generator with $\hat{f}$. In fact, this generator is a component in our construction, and so we do not really need a "base function". (In the formal proof we use a "bogus" base function that is constant so that we can apply Lemma 3.3). We first restate Theorem 1.4 in the more general terminology of pseudo-random generator schemes:

Theorem 4.1. *For every $k < n$ there is an explicit $(k, 1/m)$-pseudo-random generator scheme $E : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^m$ with $m = k^{\Omega(1/r)}$ for $r = \log\log \frac{\log n}{\log m}$.*

Proof. Given n and k let $l = \log n$ and c be the constant from Lemma 3.3 and choose m to be the largest integer so that $m^{cr+6} \leq k$. Note that $m = k^{\Omega(1/r)}$. We use a "bogus" $Base$ function. Let $Base : \{0,1\}^{m^6} \times \{0,1\}^0 \to \{0,1\}^m$ be the constant zero function. As any function over $6 \log m$ bits has a circuit of size $k_{Base} = m^6$ it follows trivially that $Base$ is a $(k_{Base}, 1/2m)$-pseudo-random generator scheme. Applying Lemma 3.3 we get that $Rec : \{0,1\}^n \times \{0,1\}^{O(l)} \to \{0,1\}^m$ is $(k_{Base} m^{cr}, 1/m)$-pseudo-random generator scheme as required. $\square$

Theorem 1.4 immediately follows from Theorem 4.1 and Lemma 2.5.

4.2 Extractors with large error

In this Section we apply our technique to construct extractors. Our technique is tailored towards constructing extractors with error $\epsilon = 1/m$. We explain how to get lower error in the next section.

Achieving seed length $O(\log n)$: We use Trevisan's extractor as Base:

Theorem 4.2. *[Tre01] For every $\delta > 0$ there is an explicitly constructible family of $(k, 1/m)$-extractors $Ext_{n,k} : \{0,1\}^n \times \{0,1\}^{O(\frac{log^2 n}{\log k})} \to \{0,1\}^{k^{1-\delta}}$.*

Applying Lemma 3.5 we obtain the following Corollary:

Corollary 4.3. *For every $\delta > 0$ there is an explicitly constructible family of $(k, O(1/m))$-extractors $Ext_{n,k} : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^{k^{1-\delta}}$.*

Proof. (of Corollary 4.3) We use the extractor of [Tre01] (Theorem 4.2) as $Base$. Given a constant δ, we pick $k_{Base} = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}}$. Indeed, the extractor of [Tre01] is an explicit $(k_{Base}, 1/2m)$-extractor $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$, where $d_{Base} = O(\log m)$. Using Lemma 3.5 we get that Rec is an explicit $(k, 1/m)$-extractor, for $k = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}} + crm < m^{\frac{1}{1-\delta}}$. The seed length of Rec is $O(\log n) + d_{Base} = O(\log n)$. □

Extracting a larger fraction of the randomness: We use the extractor of Raz, Reingold and Vadhan as Base:

Theorem 4.4. *[RRV02] For every $\delta > 0$ there is an explicitly constructible family of $(k, 1/m)$-extractors $Ext_{n,k} : \{0,1\}^n \times \{0,1\}^{O(\log^2 n)} \to \{0,1\}^{(1-\delta)k}$.*

Applying Lemma 3.5 we obtain the following Corollary:

Corollary 4.5. *There is an explicitly constructible family of $(k, O(1/m))$-extractors $Ext_{n,k} : \{0,1\}^n \times \{0,1\}^{O(\log n + \log^2 k)} \to \{0,1\}^m$, for $m = \Omega(\frac{k}{\log\log(\log n/\log k)})$.*

Proof. (of Corollary 4.5) We use the extractor of [RRV02] (Theorem 4.4) as $Base$. We pick $k_{Base} = 2m$. Indeed, the extractor of [RRV02] is an explicit $(k_{Base}, 1/2m)$-extractor $Base : \{0,1\}^{m^6} \times \{0,1\}^{d_{Base}} \to \{0,1\}^m$, where $d_{Base} = O(\log^2 m)$. Using Lemma 3.5 we get that Rec is an explicit $(k, 1/m)$-extractor, for $k = 2m + O(rm) = O(m \log\log(\log n/\log k))$ as required. The seed length of Rec is $O(\log n) + d_{Base} = O(\log n + \log^2 k)$. □

4.3 Extractors with small error

Our technique yields extractors with rather large error, ($\epsilon = O(1/m)$). In [RRV99] it was shown how to transform an extractor with large error into one that works for arbitrary small error ϵ. When one starts with error $\epsilon = O(1/m)$, this transformation increases the seed length only by an inevitable and optimal additive factor of $O(\log \frac{1}{\epsilon})$. The transformation of [RRV99] slightly hurts the output length. The extractor one gets only extracts a constant fraction of the randomness extracted by the initial extractor. This does not matter in our case since we lose a constant fraction of the randomness anyway. We now formally state the transformation of [RRV99].

Theorem 4.6. *[RRV99] Every $(k, O(1/m))$-extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ can be transformed into a $(k+O(\log(1/\epsilon), \epsilon)$-extractor $E' : \{0,1\}^n \times \{0,1\}^{d+O(\log(1/\epsilon))} \to \{0,1\}^{\Omega(m)-O(\log(1/\epsilon)}$ as long as $\epsilon > exp(-n/(\log^* n)^{\log^* n})$. Furthermore, if E is an explicit extractor then E' is an explicit extractor.*

Performing this transformation on Corollaries 4.3,4.5 gives Theorems 1.5,1.6.

5 Notions of optimality for pseudo-random generators constructions

In this section we examine two notions of optimality for pseudo-random generators constructions, and show that our construction comes close to optimal in both notions.

5.1 Pseudo-random generator schemes give extractors

Our construction of pseudo-random generators gives a $(k, 1/m)$-pseudo-random generator scheme $G : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^m$ for every $\log n < k < n$ and $m = k^{\Omega(1/\log\log(\frac{\log n}{k}))}$. We now show that the seed length in this construction is optimal except for the exact constant by giving a lower bound on the seed length of every pseudo-random generator scheme.

Lemma 5.1. *For every $\log n < k < n$, every $(k, 1/3)$-pseudo-random generator scheme with $m > d$ has $d = \Omega(\log n)$.*

Proof. By Lemma 2.7 a $(k, 1/3)$-pseudo-random generator scheme is a $(O(k \log k), 1/3)$-extractor scheme. By Theorem 2.8 every extractor scheme is an extractor. It was shown by [NZ96, RTS00] that every extractor with $m > d$ has $d = \Omega(\log n)$. $\square$

5.2 Pseudo-random generators imply hard functions

In this section, we show how to convert a pseudo-random generator into a hard function. It follows that a pseudo-random generator construction which does significantly better than ours also gives a harder function than the one initially supplied to it.

Lemma 5.2. *If there exists a pseudo-random generator $G : \{0,1\}^{l-1} \to \{0,1\}^{m(l)}$ that is computable in time $2^{O(l)}$ and $1/2$-fools all $m(l)$ size circuits then there exists a function $f = \{f_l\}$, $f_l : \{0,1\}^l \to \{0,1\}$ that is computable in time $2^{O(l)}$ so that for any l, f_l cannot be computed by circuits of size $m(l)$.*

We remark that the same conclusion holds even if G is a "hitting set generator", (see [ISW99]).

Proof. (of Lemma 5.2) We define a function $f = \{f_l\}$ $f_l : \{0,1\}^l \to \{0,1\}$. Given an l bit input x, we define $f_l(x)$ to be 1 if and only if there exist some $z \in \{0,1\}^{m(l)-l}$ for which $x \circ z$ is an output of the generator G on $l-1$ bit long inputs. We claim that $S(f_l) > m(l)$ since any circuit C computing f is not $1/2$-fooled by G. This is because C always answers 1 when given the output of the generator, and answers 1 with probability at most $1/2$ when given a uniformly chosen input. (Note that there are at most 2^{l-1} possible outputs of G on $l-1$ bit long inputs). f can be computed in time $2^{O(l)}$ by generating all outputs of the generator G on $l-1$ bit strings. $\square$

6 Conclusions, open problems and subsequent work

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 length of our pseudo-random generator and that of the best expected generator. (Namely, we get $m = k^{\Omega(\frac{1}{\log\log(l/\log k)})}$ whereas we expect to get $m = k^{\Omega(1)}$). In a subsequent work, [SU01, Uma02] give a new construction of pseudo-random generator schemes which does not use the Nisan-Wigderson generator and achieves $m = k^{\Omega(1)}$.

There's also a gap between the output length of our extractor construction and the optimal extractor, here the gap is quantitively smaller (since using Kolmogorov complexity we get $m = k^{1-\delta}$). However we are more picky in the parameters when it comes to extractors and the optimal extractor achieves $m = k + d - 2\log(1/\epsilon) - O(1)$. In subsequent works [RSW00, TSUZ01, SU01, LRVW03] this gap is made smaller, however existing constructions do not achieve the optimal output length with seed length $O(\log n)$. The reader is referred to a survey on extractor constructions [Sha02].

Our construction works by reducing the problem of constructing extractors for general sources to that of constructing extractors for sources with polynomial min-entropy. In retrospect, this transformation can be seen as a *condenser* that is a function $Con : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^{k^{O(1)}}$ such that for any distribution P with min-entropy at least k the output distribution $Con(P, U_{O(\log n)})$ is (close to) a distribution with min-entropy very close to k. (The final step towards constructing an extractor is running some prespecified "base extractor" on the output distribution of the condenser). A related approach was used in the subsequent [TSUZ01] to construct "lossless condensers" in which preserve all the min-entropy in the source distribution P.

Acknowledgments

We thank Oded Goldreich for a conversation[12] that started us working on this paper, and for many helpful comments. We are grateful to anonymous referees for many helpful comments that improved the presentation.

References

[ACR96] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. Hitting sets derandomize BPP. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium*, volume 1099 of *Lecture Notes in Computer Science*, pages 357–368, Paderborn, Germany, 8–12 July 1996. Springer-Verlag.

[ACRT99] Andreev, Clementi, Rolim, and Trevisan. Weak random sources, hitting sets, and BPP simulations. *SICOMP: SIAM Journal on Computing*, 28, 1999.

[12] which went roughly like this:
Oded: Does the IW97 technique implies generators with optimal seed length for every hardness assumption?
Avi: Obviously!
Oded: How?
Avi: Ummmm...

[BFNW93] 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.

[Blu86] M. Blum. Independent unbiased coin flips from a correlated biased source–A finite state markov chain. *COMBINAT: Combinatorica*, 6, 1986.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[CG89] B. Chor and O. Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, March 1989.

[CNS99] J.Y. 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.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Gol98] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag, Algorithms and Combinatorics, 1998.

[Imp95] 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 October 1995. IEEE.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, pages 181–190, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the thirty second annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, [2000]*, pages 1–10, 2000.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[IW01] R. Impagliazzo and A. Wigderson. Randomness vs time: Derandomization under a uniform assumption. *JCSS: Journal of Computer and System Sciences*, 63, 2001.

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

[LRVW03] C. J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: optimal up to constant factors. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 602–611, 2003.

[Nis96] 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.

[NTS99] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

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

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, February 1996.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Error reduction for extractors. In *40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, pages 191–201, 1999.

[RRV02] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. *JCSS: Journal of Computer and System Sciences*, 65, 2002.

[RSW00] O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness via repeated condensing. In *41st Annual Symposium on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California*, pages 22–31, 2000.

[RTS00] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, February 2000.

[Sha02] R. Shaltiel. Recent developments in extractors. *Bulletin of the European Association for Theoretical Computer Science*, 77, 2002.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *42nd IEEE Symposium on Foundations of Computer Science: proceedings: October 14–17, 2001, Las Vegas, Nevada, USA*, pages 648–657, 2001.

[SV86] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, April 1986.

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

[TSUZ01] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing: Hersonissos, Crete, Greece, July 6–8, 2001*, pages 143–152, 2001.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 627–634, 2002.

[WZ93] 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.

[Yao82] 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 November 1982. IEEE.

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