Near-Optimal conversion of Hardness into Pseudo-Randomness

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
avi@cs.huji.ac.il

Abstract

Various efforts ([3, 5, 6, 9]) have been made in recent years to derandomize probabilistic algorithms using the complexity theoretic assumption that there exists a problem in $E = dtime(2^{O(n)})$, that requires circuits of size $s(n)$, (for some function s). These results are based on the NW-generator [7]. For the strong lower bound $s(n) = 2^{\epsilon n}$, [6], and later [9] get the optimal derandomization, $P = BPP$. However, for weaker lower bound functions $s(n)$, these constructions fall far short of the natural conjecture for optimal derandomization, namely that $bptime(t) \subseteq dtime(2^{O(s^{-1}(t))})$. The gap in these constructions is due to an inherent limitation on efficiency in NW-style pseudo-random generators.

In this paper we are able to get derandomization in almost optimal time using any *lower bound $s(n)$. We do this by using the NW-generator in a new, more sophisticated way. We view any failure of the generator as a reduction from the given "hard" function to its restrictions on smaller input sizes. Thus, either the original construction works (almost) optimally, or one of the restricted functions is (almost) as hard as the original. Any such restriction can then be plugged into the NW-generator recursively. This process generates many "candidate" generators, and at least one is guaranteed to be "good". Then, to perform the approximation of the acceptance probability of the given circuit, we use ideas from [1]: we run a tournament between the "candidate" generators which yields an accurate estimate.*

Following Trevisan, we explore information theoretic analogs of our new construction. Trevisan [11] (and then [8]) used the NW-generator to construct efficient extractors. However, the inherent limitation of the NW-generator mentioned above makes the extra randomness required by that extractor suboptimal (for certain parameters). Applying our construction, we get an almost optimal disperser.

1. Introduction

This paper addresses the question of hardness versus randomness trade-offs. Such results [12, 3, 5, 6, 9] show that probabilistic algorithms can be efficiently simulated deterministically under some complexity theoretic assumptions. A number of such results are known under a "worst-case circuit complexity assumption":

The s-worst-case circuit complexity assumption: There exists a function $f = \{f_n\}$ which is computable in time $2^{O(n)}$, yet for all n, circuits of size $s(n)$ cannot compute f_n.

The conclusion we are after is of the following type: "Any probabilistic algorithm that runs in time t, can be simulated deterministically in time $T(t)$". Such results are usually called "Hardness Versus Randomness tradeoffs", since they trade between the existence of hard functions and the

*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

power of randomized computation. Examples of hardness vs. randomness tradeoffs based on worst-case circuit complexity assumptions may be found in [3, 5, 6, 9], (see also table 1). Our contribution is a construction that gives a better tradeoff between the simulation quality T and the strength of the assumed lower bound $s(n)$. The improvement is especially noticeable for "mid-level" strength functions $s(n)$, for example $s(n) = 2^{n^{\epsilon}}$.

1.1. Derandomization via generators

Following [12], the task of derandomizing (two sided error) probabilistic algorithms reduces to the problem of deterministically approximating the fraction of inputs which a given circuit accepts. Define an approximator as a deterministic algorithm that gets as input a circuit and approximates the fraction of inputs accepted by the circuit. An approximator can be used to deterministically simulate a probabilistic algorithm on a given input in the obvious way. The running time of the simulation is roughly the running time of the approximator, and our goal becomes constructing efficient (in terms of running time) approximators. Previous Hardness vs. Randomness tradeoffs constructed efficient approximators via pseudo-random generators.

A pseudo-random generator is a family of functions $G_n : \{0,1\}^{m(n))} \rightarrow \{0,1\}^{t(n)}$, which is computable in time $2^{O(m)}$, and has the property that for every n, the set of all outputs of G_n can be used to approximate[1] the fraction of inputs accepted by any circuit of size $t(n)$.

Intuitively, the generator G "stretches" a short seed of m random bits into a long string of t pseudo-random bits which "fool" every circuit of size $t(n)$.

A pseudo-random generator is sufficient to construct an approximator. Simply run the given circuit on all outputs of the pseudo-random generator. Thus the running time of this approximator is exponential in the generator's seed size. (One has to go over 2^m seeds and activate the generator which runs in time $2^{O(m)}$).

If one settles for derandomizing one sided error probabilistic algorithms, the object in need is a Hitting set. A hitting set H (for circuits of size t) is a (multi)-set of strings in $\{0,1\}^t$ which has the property that for every circuit C of size t which accepts at least half of the inputs there exists an element $x \in H$, which C accepts. A hitting set generator is a family of functions $G_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^{t(n)}$, which is computable in time $2^{O(m)}$, and has the property that for every n, the set of all outputs of G_n is a hitting set for circuits of size $t(n)$. Every pseudo-random generator is a hitting set generator, and the converse is not necessarily true.

In either one-sided or two-sided simulations, the running time of the simulation is exponential in the seed length. Therefore to achieve good simulations one must build generators with short seeds.

In the rest of the paper we omit the term "pseudo-random" when referring to pseudo-random generators.

1.2. Hardness Amplification

Nisan and Wigderson [7] construct a generator making use of the following distributional circuit complexity assumption, which is seemingly stronger than the worst-case circuit complexity assumptions above:

The v-distributional complexity hardness assumption: There exists a function $h = \{h_n\}$ which is computable in time $2^{O(n)}$, yet for all n, every circuit of size $v(n)$ computes h_n correctly on at most $1/2 + 1/v(n)$ fraction of the inputs.

Intuitively, the distributional complexity hardness assumption, assumes not only that the function is hard to compute, but also that it is hard to compute even "on average". Nisan and Wigderson started from the v-distributional complexity hardness assumption, and constructed a pseudo random generator which takes $m = O(\frac{n^2}{\log v(n)})$ bits, and produces $t = v(n)^{\Omega(1)}$ bits.

Previous results using worst-case assumptions ([3, 5, 6, 9]) focused on "hardness amplification", that is showing the v-distributional complexity hardness assumption follows from the s-worst case circuit complexity assumption (with some relation between s and v). Hardness vs. randomness tradeoffs then follow using hardness amplification and activating the NW-generator. Recently, [9] came up with an almost optimal hardness amplification scheme. Informally speaking, they show that the two assumptions are equivalent. More precisely, they show that given a function $f : \{0,1\}^n \rightarrow \{0,1\}$ that meets the s-worst case circuit complexity assumption, one can construct a function $h = \{h_n\}$ which meets the requirements of the $v = s^{\Omega(1)}$-distributional complexity hardness assumption. This is optimal for our purposes since we are indifferent to the difference between s and $s^{\Omega(1)}$.

Having pushed the hardness amplification phase to the limit, any remaining inefficiency in the derandomization process is caused by the NW-generator. Indeed, there are some inherent limits to the NW-generator that make these derandomization sub-optimal for functions $s(n)$ which are not exponential.

When assuming the s-worst case circuit complexity assumption, one may hope to get a generator

[1] Formally, we want that for some function $\epsilon(n)$, (which measures the quality of the approximation), and for any circuit C of size $t(n)$: $|Pr_{y \in_R \{0,1\}^{t(n)}}(C(y) = 1) - Pr_{x \in \{0,1\}^{m(n)}}(C(G_n(y)) = 1)| \leq \epsilon$ The circuit C may take less than $t(n)$ bits as input. We don't want to keep a special parameter for the input size. Instead, we prepare in advance $t(n)$ bits, and if C is on less bits, it reads only a prefix of them

Table 1. Result Comparison

All results assume the s-worst-case circuit complexity assumption.

Reference	Conclusion for arbitrary $s(n)$	Conclusion for $s(n) = 2^{n^\epsilon}$
[3]	$bptime(t) \subseteq dtime(2^{O((s^{-1}(t^{O(1)}))^4 \log^3 t)})$	$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{4}{\epsilon}+1} \cdot (\log\log t)^2)})$
[5]	$bptime(t) \subseteq dtime(2^{O((s^{-1}(t^{O(1)}))^2 \log t)})$	$bptime(t) \subseteq dtime(2^{O(\log^{\frac{2}{\epsilon}+1} t)})$
[6][a]	$bptime(t) \subseteq dtime(2^{O(\frac{(s^{-1}(t^{O(1)}))^4}{\log^3 t})})$	$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{4}{\epsilon}-3})})$
[9]	$bptime(t) \subseteq dtime(2^{O(\frac{(s^{-1}(t^{O(1)}))^2}{\log t})})$	$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{2}{\epsilon}-1})})$
this paper[b]	$bptime(t) \subseteq dtime(2^{O(s^{-1}(t^{O(\log\log t)}))})$	$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{1}{\epsilon}} \cdot \log\log\log t)})$
optimal[c]	$bptime(t) \subseteq dtime(2^{O(s^{-1}(t))})$	$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{1}{\epsilon}})})$

[a] Impagliazzo and Wigderson state their result only for $s(n) = 2^{\Omega(n)}$, and their result puts BPP in P, for such a lower bound.
[b] Our result is a bit better, but we cannot state it in this notation.
[c] The best we can hope for with current techniques, see section 6.

$$OPT : \{0,1\}^{O(n)} \to \{0,1\}^{s(n)^{\Omega(1)}}$$

that fools circuits of size $s(n)^{\Omega(1)}$, (see section 6). However, the best result using the NW-generator takes a larger seed:

$$NW : \{0,1\}^{O(\frac{n^2}{\log s(n)})} \to \{0,1\}^{s(n)^{\Omega(1)}}$$

When $s(n) = 2^{\Omega(n)}$ one gets the desired $m = O(n)$. For weaker lower bounds the seed size becomes larger.

1.3. Our result: overview and intuition

In this paper we are able to minimize the seed size to the optimal $m = O(n)$. However, we do lose something. The first loss is that we are only able to "fool" size $t = s(n)^{\Omega(\frac{1}{\log\log n})}$ circuits rather than size $s(n)^{\Omega(1)}$ circuits[2]. The second loss is that rather than constructing a generator, we construct $2^{O(n)}$ candidate generators where at least one of them is a "good" generator. We don't know how to find the good generator in the huge collection. Nevertheless, a trivial observation is that this collection may be used to construct a hitting set generator. This is because the set of outputs of all candidate generators is a small hitting set. This suffices for one sided error probabilistic algorithms. As for two sided error probabilistic algorithms, we show that this collection of candidate generators is useful to construct an approximator which runs in time $2^{O(n)}$, (that is as efficient as the one we would have built had we been able to locate the good generator).

[2] This means that in order to derandomize a probabilistic algorithm that runs in time t, we need $n \geq (s^{-1}(t^{O(\log\log n)}))$ rather than $n \geq s^{-1}(t^{O(1)})$. While this is not optimal, it is much better than previous constructions.

To explain the inherent inefficiency in the NW construction, and how we overcome it, we briefly describe it.

The NW-generator: As mentioned before, to use the NW-generator one needs a (distributional complexity) hard function h. Using the optimal hardness amplification of [9], we may assume that this function is $s(n)^{\Omega(1)}$-distributional complexity hard. Given numbers m and t the NW-construction gives a function $NW_h : \{0,1\}^m \to \{0,1\}^t$, and a number $k = \Theta(\frac{n^2}{m})$. The main lemma of [7] says that if the NW-generator is not a pseudo-random generator then h is easy. More precisely, the statement is that if there exists a circuit of size t which "catches" NW_h, then there exists a circuit of size $t2^k$ which approximates h. Put differently, this means that when used with seed size m, the NW-generator can fool circuits of size $t = s^{\Omega(1)}/2^k$. To get $t = s^{\Omega(1)}$, one needs $k = \Theta(\log s)$. Since $k = \Omega(\frac{n^2}{m})$, the seed size must be increased to $m = \Omega(n^2/\log s)$.

This gives a tradeoff between m (the seed size) and tt (he size of the circuits we want to fool). On one hand we must have k small, because a factor of 2^k is lost when getting t from s. On the other, small k forces large m since $m = \Omega(n^2/k)$.

The new idea: We want to use the NW-generator with seed size $m = O(n)$. As mentioned earlier this forces $k = \Omega(n)$, and if s is not exponential, we cannot afford the factor of 2^k lost in the circuit complexity by the NW-lemma. In other words if $s = 2^{o(n)}$, there is no guarantee that using the NW-generator with these parameters we get a good generator.

The main idea behind this paper is that the 2^k factor lost in the NW-lemma is caused by a pessimistic bound on circuit complexity of some functions on k bits. We distinguish between two cases: The optimistic case is that these functions can be computed by much smaller circuits. In this case the NW-lemma shows that NW_h is a good generator, and we don't lose the 2^k factor. In the second case the pessimistic bound applies and the NW-construction may fail to produce a correct generator. However, in this case we get a function which is harder than the original one we started with. This puts us in better condition as the NW-generator works better if you have a harder function.

To be more precise, we make the following observation: when used with some function h the NW-generator specifies a family of $2^{O(n)}$ functions over k bits (which are restrictions of the given function h). Either NW_h is not a good generator, or one of the specified functions requires large circuit size. [7] choose k small enough to ensure that the latter case is impossible using the fact that every function on k bits has a circuit of size at most 2^k. We instead choose $k = \Omega(n)$ and consider the following two cases:

1. All the functions specified by the NW-generator have "small" (size $s/t^{O(1)}$) circuits. In such a case we get that the NW generator is indeed good for circuits of size t, and we don't lose the 2^k factor.
2. At least one of the specified functions cannot be computed by a circuit of size $s/t^{O(1)}$. In this case it may be that the NW-generator is bad. However, we have at hand a function on many fewer bits than the original hard function (k instead of n), that requires roughly the same circuit size. From the point of view of the ratio between hardness to input size, this function is harder than the one we started with. We can "plug" it to the NW-generator and enjoy the better lower bound. (Recall that the NW-generator is more efficient with strong lower bounds). This approach can be used recursively until we end up with a function which requires circuits of size exponential in the length of it's input. On such a function we can afford to use the old proof.

The construction: While this may seem very appealing, there are still some problems. We don't know which of the two cases happened, and even worse, in case 2, we don't know which of the $2^{O(n)}$ specified functions is the hard function. Thus, we try all possibilities. We construct candidate generators from the initial function and all its specified functions. We continue this recursively until we are sure that one of the functions we consider is "hard" but all its specified functions are "easy". This can be shown to happen after at most $O(\log \log n)$ recursion levels, and at this point we have $2^{O(n)}$ candidates. This process involves some loss[3]. At this point we know that at least one candidate is a good generator.

[3]At each level we lose a fraction in the circuit complexity, and this is the reason end up with $t = s^{\Theta(\frac{1}{\log \log n})}$.

From here we may continue in two different paths. It is easy to see that the union of outputs of all candidate generators is a hitting set of size $2^{O(n)}$ for circuits of size t. This makes our construction a hitting set generator and suffices for one sided error probabilistic algorithms. We are not able to spot the good generator. However, in order to derandomize two sided error probabilistic algorithms it is enough to construct an approximator. [1] showed how to construct an efficient approximator given a hitting set generator. Our hitting set generator has special properties that makes the proof simpler. Recall that to construct an approximator, we need to approximate the fraction of inputs accepted by a given circuit. The idea is to hold a "tournament" between all the candidate generators. The winner of this tournament is not necessarily a good generator. However, it is certain to give a good approximation of the fraction of inputs accepted by the circuit we are given as input.

A disperser a la Trevisan: Recently, Trevisan [11] used the NW-generator to construct an extractor. An extractor is an efficiently computable function $Ext : \{0,1\}^l \times \{0,1\}^m \rightarrow \{0,1\}^t$, such that for all distributions D on $\{0,1\}^l$ having min-entropy[4] s, the distribution obtained by sampling f according to D, x uniformly from $\{0,1\}^m$ and computing $Ext(f,x)$, is statistically close to the uniform distribution on t bits. Trevisan's extractor works by treating f as a function over $n = \log l$ bits, "amplifying" its hardness, and applying $NW_h(x)$.

We focus our attention to constructing extractors with minimal m. Trevisan's construction suffers from the same inefficiency of the NW-generator which we treat here. As suggested by our choice of letters, in the extractor terminology, the min-entropy takes the role of "hardness". Indeed, Trevisan extractors use $m = O(\frac{\log l}{\log s}) = O(\frac{n^2}{\log s})$ bits. As in the case of pseudo-random generators, the optimal[5] seed size is $m = O(n)$. We don't get an improved extractor, since our construction does not give a pseudo random generator. Instead we get the information theoretic analog of a hitting set generator which is called a disperser. The exact definition appears on section 5. Unlike extractors, there exists an explicit construction of optimal dispersers by [10]. Our construction is slightly inferior to the optimal one, but involves totally different methods.

The technique used in this paper (combined with some more ideas) can be used to construct an almost optimal pseudo-random generator and therefore an almost optimal extractor. We delay this to a future paper.

[4]The min-entropy of a distribution D is $-\log(min_x D(x))$.

[5]We neglect the dependence of m on the statistical distance required between the uniform distribution and the obtained distribution, and assume that the statistical distance required is an arbitrary small constant

1.4. Organization of the paper

Section 2 includes definitions and cites the previous results needed for our construction. Section 3 includes the main theorem and the main construction. Section 4 shows how to use the main construction to give an approximator. Section 5 concerns using Trevisan's method with our result and constructs a disperser. Section 6 includes a construction of hard functions from hitting set generators and explains what we mean by optimal derandomization.

2. Definitions and History

2.1. Hard functions

We start by defining "hardness" in both worst-case complexity and distributional complexity settings.

Definition 1 *For a function $f : \{0,1\}^n \rightarrow \{0,1\}$, we define:*

1. $S(f) = min\{size(C)|$ *circuits C that compute f correctly on every input* $\}$
2. $SUC_s(f) = max\{Pr_{x \in_R \{0,1\}^n}(C(x) = f(x))|$ *circuits C of size s* $\}$
3. $ADV_s(f) = 2SUC_s(f) - 1$

When invoking the NW-generator against circuits of size $t(n)$, one needs a function $h = \{h_n\}$, with $ADV_{t(n)}(h_n) \leq \frac{1}{t(n)}$. Much work has been done on building such a function from a worst-case circuit lower bound, (see for example [12, 3, 5, 6]). In this paper we use the best known result by [9], which we state in our notation.

Theorem 1 *[9] For every function $f : \{0,1\}^n \rightarrow \{0,1\}$, and ϵ, there exists a function $h : \{0,1\}^{4n} \rightarrow \{0,1\}$, such that:*

1. *h can be computed in time $2^{O(n)}$, given an oracle to f.*
2. *$ADV_v(h) \leq \epsilon$, for $v = S(f)(\frac{\epsilon}{n})^{O(1)}$*

2.2. Generators, Discrepancy sets, Hitting sets and Approximators

In this section we define pseudo-random/hitting set generators, and algorithms we call approximators. In the following definitions we will use the same parameter t, both for the size of the circuit, and the size of the input given to the circuit. A circuit of size t takes as input at most t bits, and in case the circuit takes less bits we assume it takes a prefix of the t bits that we prepare in advance.

Definition 2 *For a circuit C of size t bits define: $\mu(C) = Pr_{w \in_R \{0,1\}^t}(C(w) = 1)$*

Definition 3 *A (t, ϵ)-discrepancy set, is a multi-set $D \subseteq \{0,1\}^t$, such that for all circuits C of size t:*

$$|Pr_{w \in_R D}(C(w) = 1) - \mu(C)| \leq \epsilon$$

Definition 4 *A (t, ϵ)-hitting set, is a multi-set $H \subseteq \{0,1\}^t$, such that for all circuits C of size t, if $\mu(C) > \epsilon$ then there exists $w \in H$ such that $C(w) = 1$.*

An easy observation is that a discrepancy set is also a hitting set, while the converse need not be true. We proceed and define pseudo-random/hitting set generators. In both cases, for the purpose of derandomizing probabilistic algorithms, generators may be allowed to run in time exponential in their input.

Definition 5 *A ϵ-pseudo-random generator (resp. ϵ-hitting set generator) $G = \{G_n\}$ is a family of functions $G_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^{t(n)}$, such that:*

1. *For all n, the set*

$$D = \{G_n(x)|x \in \{0,1\}^{m(n)}\}$$

   is a $(t(n), \epsilon(n))$-discrepancy set (resp. $(t(n), \epsilon(n))$-hitting set).

2. *G is computable in time $2^{O(m)}$, (exponential in the size of the input).*

From now on we will drop the "pseudo-random" when talking about pseudo-random generators.

The existence of "good" generators implies a non-trivial deterministic simulation of two sided error probabilistic algorithms. However, the proof works by building the following device (which appears implicitly since [12] and is also used implicitly in other efforts to derandomize BPP such as [1]).

Definition 6 *A ϵ-approximator is a deterministic algorithm that takes as input a circuit C and outputs an approximation of $\mu(C)$, that is, a number q such that $|\mu(C) - q| \leq \epsilon$*

The following two implications are standard:

Lemma 1 *([12])*

1. *If there exists a ϵ-generator $G : \{0,1\}^{m(n)} \rightarrow \{0,1\}^{t(n)}$, then there exists a ϵ-approximator that (on a circuit of size $t(n)$) runs in time $2^{O(m(n))}t(n)^{O(1)}$.*
2. *If there exists a $1/10$-approximator that runs in time $p(t)$ on circuits of size t, then $bptime(t) \subseteq dtime(p(t^2))$.*

Proof: (sketch)
Having a generator, one can run the given circuit on all possible outputs of the generator. This is indeed an efficient approximator. Having an approximator, and given a probabilistic algorithm $M(x, y)$, (where x is the input, and y is the random string), simply construct the circuit $C_x(y) = M(x, y)$, and approximate it's success probability. $\bullet$

As seen from lemma 1, the task of derandomizing two sided error probabilistic algorithms, reduces to constructing efficient generators, (Where efficiency means small as possible seed size). A similar argument shows that efficient hitting set generators suffice for one sided error probabilistic algorithms.

2.3. The NW-generator

In this section we present the NW-generator, it's best known consequences for derandomization, and explain it's inherent inefficiency when used with a sub-exponential lower bound.

Theorem 2 *[7] (Construction of nearly disjoint sets) There exists an algorithm that given numbers n, m, t, such that $t = 2^{O(\frac{n^2}{m})}$, constructs a (n, m)-**design**, that is: sets $S_1, .., S_t$, such that:*

1. *For all $1 \le i \le t$, $S_i \subseteq [m]$, and $|S_i| = n$.*
2. *For all $1 \le i < j \le t$, $|S_i \cap S_j| \le k = c\frac{n^2}{m}$, for some constant c.*
3. *The running time of the algorithm is exponential in m.*

Definition 7 *(The NW-generator [7]) Given some function $h = \{h_n\}$, and n, m, the NW-generator works by building an (n, m)-design, $S_1, .., S_t$. It takes as input m bits, and outputs t bits.*

$$NW_h^{n,m}(x) = (h(x|_{S_1}), .., h(x|_{S_t}))$$

The thing to do now, is prove that if one "plugs" a hard enough h into the NW-generator, it fools circuits of some size.

Lemma 2 *[7] Fix n, m, v, ϵ, and $t < 2^{O(\frac{n^2}{m})}$. Let $S_1, .., S_t$ be the (n, m)-design promised by theorem 2, and let $k = c\frac{n^2}{m}$, be the promised bound on the intersection size. Let $h : \{0,1\}^n \to \{0,1\}$, be a function such that $ADV_v(h) \le \frac{\epsilon 2^{k+1}}{v}$. The set:*

$$D = \{NW_h^{n,m}(x) |\ x \in \{0,1\}^m\}$$

is a (t', ϵ)-discrepancy set, with $t' = min(t, \frac{v}{2^{k+1}})$.

The drawback in lemma 2, is that $t = O(\frac{v}{2^k})$, and a factor of 2^k is lost when getting t from v. To cope with this k must be decreased, (particularly k must satisfy $2^k < v$). Since k is roughly $\frac{n^2}{m}$, m must be increased to roughly $\frac{n^2}{\log v}$, resulting in a generator that takes a large seed for weak lower bounds v, and an non-efficient approximator. (Recall that the running time of an approximator is exponential in the generator's seed size).

Using theorems 1,2 and lemma 2 with the parameters described above [9] prove the following theorem.

Theorem 3 *([9]) If there exists a function $f = \{f_n\}$ that is computable in time $2^{O(n)}$ and for all n, $S(f_n) \ge s(n)$, then there exists a $s^{-\Omega(1)}$-generator, $G : \{0,1\}^{O(\frac{n^2}{\log s})} \to \{0,1\}^t$, which fools circuits of size t, for $t = s^{\Omega(1)}$.*

We may still expect to have an optimal generator, that is $G : \{0,1\}^{O(n)} \to \{0,1\}^{s^{\Omega(1)}}$ which fools circuits of size $s^{\Omega(1)}$. This cannot be achieved by improving the design, as the next lemma shows that the current construction of designs is optimal.

Lemma 3 *If $S_1, .., S_t \subseteq [m]$, and for all $1 \le i \le t$, $|S_i| = n$, and for all $1 \le i < j \le t$, $|S_1 \cap S_j| \le k$, and $t \ge \frac{n}{2k}$, then $m \ge \frac{n^2}{4k}$*

Proof: It is enough to prove the lemma for $t = \frac{n}{2k}$. Using the first two terms in the inclusion-exclusion formula we get that:

$$m \ge | \cup_{1 \le i \le t} S_i| \ge \sum_{1 \le i \le t} |S_i| - \sum_{1 \le i < j \le t} |S_i \cap S_j|$$

Which is the required bound, for our choice of parameters. $\bullet$

3. The new approximator

The main result of this paper is a construction of an approximator that "corresponds" to an almost optimal generator:

Theorem 4 *If $f = \{f_n\}$ is a function computable in time $2^{O(n)}$, such that for all n, $S(f_n) \ge s(n)$, then there exists a ϵ-approximator, such that on circuits of size $s(n)^{\Omega(\frac{1}{\log \log n})}$ runs in time $2^{O(n)}$, with $\epsilon = s^{-\Omega(\frac{1}{\log \log n})}$.*

A particularly interesting case is $s(n) = 2^{n^\epsilon}$, for which we obtain:

$$bptime(t) \subseteq dtime(2^{O((\log t)^{\frac{1}{\epsilon}} \log \log \log t})$$

For more general functions $s(n)$, the above equation does not seem to have a nice closed-form solution. However, since the derandomization takes time 2^n, we can pick $n = \min\{t, s^{-1}(t^{O(\log \log t)})\}$. Then since $n \le t$, $t^{\log \log t} > t^{\log \log n}$. This gives:

Theorem 5 *Let $f = \{f_n\}$ be a function computable in time $2^{O(n)}$, such that for all n, $S(f_n) \geq s(n)$, then $bptime(t) \subseteq dtime(2^{s^{-1}(t^{O(\log\log t)})})$.*

This approximator should be compared to the one of [9], which is constructed by applying theorem 3 and lemma 1 in sequence. [9]'s approximator runs in time $2^{O(\frac{n^2}{\log s})}$ on circuits of size $s^{\Omega(1)}$.

3.1. A new lemma

In this paper we replace lemma 2, by a new lemma, saying that: "either we can build a discrepancy set for a large t, or we have at our disposal a hard function on smaller input". We could plug this function into the NW-generator, and since it's input size n is decreased, we will be able to build designs with smaller k.

Definition 8 *Given a function $h : \{0,1\}^n \to \{0,1\}$, two sets $S_1, S_2 \subseteq [m]$, where: $|S_1| = |S_2| = n$, $k = |S_1 \cap S_2|$, and $\alpha \in \{0,1\}^{n-k}$, we define a function $b_h^{S_1,S_2,\alpha} : \{0,1\}^k \to \{0,1\}$, in the following way:*

$$b_h^{S_1,S_2,\alpha}(z) = h(z;\alpha)$$

We think of S_1, as the n input bits to h. z is placed in the bits which correspond to $S_1 \cap S_2$, and α is used to "fill" the remaining $n-k$ bits.

The definition depends only on α, and the way S_2 intersects S_1. Therefore, there are at most $2^{n-k} \cdot 2^n = 2^{O(n)}$ such functions.

The exact definition of the above function is less important. The important thing is that the number of such functions (for a particular h) is bounded, and all can be easily computed if one can compute h. The following lemma replaces lemma 2 and connects the quality of the generator to the hardness of the defined functions. Its proof is directly analogous to that of lemma 2.

Lemma 4 *Fix n, m, v, ϵ and $t < min(2v, 2^{O(\frac{n^2}{m})})$. Let $S_1, .., S_t$ be the (n,m)-design promised by theorem 2, and let $k = c\frac{n^2}{m}$ be the promised bound on the intersection size. Let $h : \{0,1\}^n \to \{0,1\}$, be a function such that $ADV_v(h) \leq \frac{\epsilon}{t}$. Consider the set:*

$$D = \{NW_h^{n,m}(x) | \; x \in \{0,1\}^m\}$$

If D is not a (t,ϵ)-discrepancy set then there exist some $1 \leq i, j \leq t$, and a partial assignment $\alpha \in \{0,1\}^{n-k}$, such that: $S(b_h^{S_i,S_j,\alpha}) \geq \frac{v}{2t}$.

Remark 1 *It is worthwhile to notice that lemma 2 indeed follows from lemma 4. Simply take $t = \frac{v}{2^{k+1}}$. This matches the assumption about h. None of the restricted functions can require circuit complexity $\frac{v}{2t} \geq 2^k$, since they are functions over k bits. So it must be the case that D is a (t,ϵ)-discrepancy set.*

Proof: (Of lemma 4)
If D is not a (t,ϵ)-discrepancy set, then there exists a circuit A of size t such that:

$$|Pr_{w \in_R D}(A(w) = 1) - \mu(A)| > \epsilon$$

By using a standard "hybrid argument" as in [12], we get that there exists a circuit C of size t, and $1 \leq j \leq t$, such that C predicts the j'th bit of the generator's output from the previous $j-1$ of the bits, namely:

$$Pr_{w \in_R D}(C(w_1, .., w_{j-1}) = w_j) > \frac{1}{2} + \frac{\epsilon}{t}$$

Choosing a random w in D, amounts to choosing a random x in $\{0,1\}^m$, and applying $NW_h^{n,m}$. This means that w_j is nothing but $h(x|_{S_j})$. We get that:

$$Pr_{x \in_R \{0,1\}^m}(C(NW_h^{n,m}(x)|_{1..j-1}) = h(x|_{S_j})) > \frac{1}{2} + \frac{\epsilon}{t}$$

There exists a fixing $\beta \in \{0,1\}^{m-n}$ to the bits outside of S_j, such that:

$$Pr_{y \in_R \{0,1\}^n}(C(NW_h^{n,m}(y;\beta)|_{1..j-1}) = h(y)) > \frac{1}{2} + \frac{\epsilon}{t} \qquad (1)$$

For $i < j$, if we set $\alpha_i = \beta|_{S_i \setminus S_j}$, we get:

$$NW_h^{n,m}(y;\beta)|_i = h((y;\beta)|_{S_i}) = b_h^{S_i,S_j,\alpha_i}((y;\beta)|_{S_i \cap S_j})$$

Therefore, if it is the case that all the $b_h^{S_i,S_j,\alpha}$'s had low circuit complexity, then there are size $\frac{v}{2t}$ circuits which compute $NW_h^{n,m}(y;\beta)|_i$ for all $i < j$. Combining these with C, and using (1), we get a circuit D of size $t + t \cdot \frac{v}{2t} \leq v$, such that:

$$Pr_{y \in_R \{0,1\}^n}(D(y) = h(y)) > \frac{1}{2} + \frac{\epsilon}{t}$$

Which is a contradiction. •

3.2. The construction

In this section, we start presenting the new approximator. The first step will be designing an algorithm that takes a circuit size as input, and constructs a collection of small sets, where at least one of them is a "good" discrepancy set. In the next section we deal with the problem of approximating the fraction of the inputs accepted by a given circuit with such a collection.

To build the generator we need a function $f = \{f_n\}$, such that:

1. f is computable in time $2^{O(n)}$.
2. For all n, $S(f_n) \geq s(n)$.

Parameters for the construction:

m	-	the seed length.
t	-	the length of the "pseudo-random" string, (which is also the size of the circuit we want to fool).
ϵ	-	a bound on the error of the generator.
n	-	an input length on which f_n is hard.
s	-	the lower bound known on f_n, (that is a number such that: $S(f_n) \geq s$).

The construction works by recursively calling the procedure **construct(**l, n, s, g**)**, (where l, n, s are integers and g is a function from $\{0,1\}^n$ to $\{0,1\}$, represented as a truth table). (m and t are also inputs, but left unchanged in recursive calls.) The first call is to **construct(**$1, n, s, f_n$**)**

construct(l, n, s, g**)**

1. Use theorem 1 to create a function $h : \{0,1\}^{4n} \rightarrow \{0,1\}$, such that if $S(g) = s$, then $ADV_v(h) \leq \frac{\epsilon}{t}$. (This can be achieved with $v = s(\frac{\epsilon}{tn})^{O(1)}$).
2. Use theorem 2 to create a $(4n, m) - design, S_1, .., S_t$.
3. Let $k = c\frac{(4n)^2}{m}$ be the bound on the intersection size.
4. Output $D = \{NW_h^{4n,m}(x) | x \in \{0,1\}^m\}$.
5. If $\frac{v}{2^{k+1}} \geq t$, return.
6. For all $i \neq j \in [t]$, and for all $\alpha \in \{0,1\}^{n-k}$, Call **construct(** $l+1, k, \frac{v}{2t}, b_h^{S_i,S_j,\alpha}$**)**. Note that it is easy to compute a truth table for $b_h^{S_i,S_j,\alpha}$ given a truth table for g.

Note that for each instantiation of **construct**, l is the level of the instantiation in the recursion tree. The values of n, s, v, k depend only on l, and so we call them n_l, s_l, v_l, k_l respectively.

Theorem 6 *There exists a constant d so that under the following assumptions:*

1. $S(f_n) \geq s$.
2. *f is computable in time* $2^{O(n)}$.
3. $t = s^{\frac{1}{d \log\log n}} > n$.
4. $\epsilon = \frac{1}{t}$.
5. $m = 2cn$.

The process described runs in time $2^{O(n)}$, and at least one of the sets D generated is a (t, ϵ)-discrepancy set.

The theorem will follow from a sequence of claims.

Claim 1 *Using the conditions of theorem 6, it is easy to get the following equations.*

- $v_l = \frac{s_l}{t^{O(1)}}$.
- $s_l = \frac{s}{t^{O(l)}}$.
- $n_l = O(\frac{n}{2^{2^{l-1}}})$

Claim 2 *The process described can be performed in time* $2^{O(n)}$.

Proof: We have already fixed $m = O(n)$. The work done in each instantiation of **construct** can be done in time $2^{O(m)} = 2^{O(n)}$. We will bound the size of the recursion tree. The degree of the recursion tree at level l is bounded by $2^{O(n_l)}$. Using the fact that for all l, $n_{l+1} \leq \frac{n_l}{2}$, we can bound the degree of the recursion tree at level l by $2^{O(n_l)} = 2^{O(\frac{n}{2^l})}$. This means that the total number of instantiations is bounded by

$$\prod_l 2^{O(\frac{n}{2^l})} = 2^{n \sum_l \frac{1}{2^l}} = 2^{O(n)}$$

•

Claim 3 *The depth of the recursion tree is bounded by* $O(\log\log n)$.

Proof: We simply have to estimate l such that $\frac{v_l}{2^{k_l}} \geq t$
Using our former equations this translates to:

$$\frac{s}{s^{O(\frac{l}{2\log\log n})} 2^{\frac{n}{2^{2^l}}}} \geq s^{\frac{1}{2\log\log n}}$$

Which is satisfied by taking $l = \Theta(\log\log n)$. •

Claim 4 *Suppose that all the sets D produced up to level $l-1$, are not (t, ϵ)-discrepancy sets, then there exists a g in level l such that $S(g) \geq s_l$.*

Proof: The proof uses induction on l. The claim is certainly true for $l = 1$. For $l > 1$, We know that in levels 1 up to l there is no (t, ϵ)-discrepancy set. Using the induction hypothesis for levels 1 up to $l-1$, we know that there is some function g_{l-1} in level $l-1$ such that $S(g_{l-1}) \geq s_{l-1}$. This means that in the same instantiation of **construct**, the function h_{l-1} had $ADV_{v_{l-1}}(h_{l-1}) \leq \frac{\epsilon}{t}$. Using lemma 4, we get that if the D produced at the current instantiation of **construct** is not a (t, ϵ)-discrepancy set, then there exists a restriction $b = b_{h_{l-1}}^{S_i,S_j,\alpha}$, (for some choice of i, j, α), such that:

$$S(b) \geq \frac{v_{l-1}}{2t} = s_l$$

•

Proof: (Of theorem 6) We have already bounded the running time in claim 2. Let d be the depth of the recursion tree. From claim 4 we get that if non of the D's in levels 1 up to d is a (t,ϵ)-discrepancy set then one of the g's in the last level has $S(g) \geq s_d$. And so, $ADV_{v_d}(h) \leq \frac{\epsilon}{t}$. At the last level we can afford the price of using lemma 2. Using it we get that D is a $(\frac{v_d}{2^{k_d+1}}, \epsilon)$-discrepancy set. Using the fact that at level d, $\frac{v_d}{2^{k_d+1}} \geq t$, we get that D is a (t,ϵ)-discrepancy set. •

4. A tournament of generators

In the previous section we constructed a collection of $2^{O(n)}$ sets, where one of them is a (t,ϵ)-discrepancy set. It is trivial to see that their union is a (t,ϵ)-hitting set of size $2^{O(n)} \cdot 2^{O(n)} = 2^{O(n)}$. This suffices to construct a hitting set generator. To complete the construction of the approximator, we need to be able to approximate the success probability of a given circuit, using such a collection. We achieve this using an idea from [1][6]. (See also [2]).

Theorem 7 *There exists an algorithm for the following computational problem:*
Input:

- *A circuit C of size t.*
- *A collection of multi-sets $D_1, .., D_l$, such that for $1 \leq i \leq l$, $D_i \subseteq \{0,1\}^t$, and $|D_i| \leq M$. Moreover, at least one of the D's is a (t,ϵ)-discrepancy set.*

Output: *A number α, such that $|\alpha - \mu(C)| \leq 2\epsilon$*
The algorithm solves the problem. and runs in time polynomial in t, l, M.

Proof: For $y \in \{0,1\}^t$, define $C_y(w) = C(w \oplus y)$. Note that for all $y \in \{0,1\}^t$, C_y is of the same size as C. For $1 \leq i \leq l$, define:

$$\alpha_i(y) = Pr_{w \in_R D_i}(C(w \oplus y) = 1)$$

For $i, j \in [l]$ define:

$$\alpha_{ij} = E_{y \in_R D_j}(\alpha_i(y))$$

Let k be an index such that D_k is a (t,ϵ)-discrepancy set. For all $y \in \{0,1\}^t$, $\mu(C) = \mu(C_y)$, and $|\alpha_k(y) - \mu(C_y)| \leq \epsilon$. From this we have that for all $1 \leq j \leq l$:

$$|\alpha_{kj} - \mu(C)| \leq \epsilon$$

Note that for all $i, j \in [l]$, $\alpha_{ij} = \alpha_{ji}$. This is because both amount to taking all pairs a_1, a_2 from D_i, D_j and running $C(a_1 \oplus a_2)$. The algorithm computes α_{ij} for all $i, j \in [l]$. and picks a row r such that all the numbers in $I = \{\alpha_{rj} | j \in [l]\}$ lie on an interval of length 2ϵ. It then returns α, the middle of the interval of I. Such an r exists, because k has that property. For all i, we have that $|\alpha_{ik} - \mu(C)| \leq \epsilon$, and therefore all the numbers in I, are at a distance of at most 3ϵ from $\mu(C)$. From this we have that $|\alpha - \mu(C)| \leq 2\epsilon$. •

By applying theorems 6,7 in sequence we get the approximator, and prove theorem 4.

[6]Actually, [1] constructs an approximator from a hitting set generator. Thus we could be done stating this result. However, we can give a much simpler proof for this particular setup.

5. An information theoretic analog a-la Trevisan

Recently, Trevisan [11] used the NW-generator to construct an extractor. Trevisan's extractor suffers from the same inefficiency of the NW-generator. In this section we use our technique to build a disperser.

Definition 9 *A function $Dis : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^t$ is called a (r,ϵ)-disperser, if for any $A \subseteq \{0,1\}^l$, such that $|A| > 2^r$, $|\Gamma(A)| > (1-\epsilon)2^t$, (where $\Gamma(A)$ denotes the set of elements y in $\{0,1\}^t$ such that there exists $a \in A$ and $x \in \{0,1\}^m$ such that $dis(a,x) = y$.*

Unlike extractors, dispersers with small seed size have already been constructed by [10]. Our construction achieves a totally different almost optimal disperser.

Theorem 8 *For every l, r there exists an $(r, t^{-O(1)})$-disperser $dis : \{0,1\}^l \times \{0,1\}^{O(\log l)} \to \{0,1\}^t$, where $t = r^{\Theta(\frac{1}{\log\log\log l})}$.*

The Trevisan argument can be used to transform the hitting set generator constructed in section 4, into a disperser. We do not give a proof, as the technique of this paper can be extended to give much better results. For example we can construct extractors which work with optimal seed size and extract a square root of the min-entropy. The construction and proof are delayed to a future paper.

6. Hitting set generators imply hard functions

We clarify our use of "optimal" in the introduction and table 1. All of the de-randomization ideas that put BPP in a deterministic time class construct hitting set generators. The following theorem shows that constructions of hitting set generators are related to lower bounds for circuit complexity of functions in $E = dtime(2^{O(n)})$.

The following is easy, but we cannot find a reference.

Theorem 9 *If there exists a $\frac{1}{2}$-hitting set generator $G = \{G_n\}$, where $G_n : \{0,1\}^{n-1} \to \{0,1\}^{t(n)}$, then there exists a function $f = \{f_n\}$, where $f_n : \{0,1\}^n \to \{0,1\}$, such that:*

1. *f is in E.*
2. *For all n, $S(f_n) > t(n)$.*

Proof: The function f is defined as follows: On input x of size n, construct the hitting set $H_n = \{G_n(y) | y \in \{0,1\}^{n-1}\}$. Accept x if it does not appear as a prefix of an element of H_n. One can easily compute f by constructing H_n, (which takes time $2^{O(n)}$), and comparing the given input to all strings in H_n. This puts f in E. Since $|H_n| = 2^{n-1}$, if one looks at the first n bits of elements in H_n, he will encounter at most half of the possible 2^n patterns. This means that h_n accepts at least half of the inputs in $\{0,1\}^n$. If h_n is computed by some circuit C of size $t(n)$, then C accepts half of the possible inputs in $\{0,1\}^n$. We may view C as a circuit that takes $t(n)$ inputs and ignores all but n of them. Since H_n is a hitting set, then there must be an element in H_n which C accepts, but this contradicts the definition of f. •

As a consequence we get that any construction that was superior to that we list as "optimal"[7], would be a proof that circuit lower bounds for functions in E could be automatically boosted to give even larger lower bounds. This would show a gap in the possible circuit complexities of functions complete for E. Since proving such a gap seems quite difficult, we believe progress in derandomization by current techniques is limited to matching the "optimal" bounds listed. Putting this in terms of simulation of probabilistic algorithms, if we start from a function with lower bound $s(n)$, the best result we can hope for is $bptime(t) \subseteq dtime(2^{s^{-1}(t)})$.

Acknowledgements

We thank Oded Goldreich for a conversation[8] that started us working on this paper, and for many helpful comments. We thank Ido Bregman for reading a preliminary version of this paper and for helpful comments.

References

[1] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. Hitting sets derandomize BPP. In F. Meyer auf der Heide and B. 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.

[2] A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. In *38th Annual Symposium on Foundations of Computer Science*, pages 264–272, Miami Beach, Florida, 20–22 Oct. 1997. IEEE.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] A. Ta-Shma. Almost optimal dispersers. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 196–202, Dallas, Texas, 23–26 May 1998.

[11] 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.

[12] 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.

---

[7] That is one that starts from a function $f = \{f_n\}$ in E, with $S(f_n) \geq s(n)$, and constructs hitting set generator that takes less than n bits to fool circuits of size $s(n)$

[8] which went roughly like this:
Oded: Does the IW97 technique implies optimal generators for every circuit lower bound assumption?
Avi: Obviously!
Oded: How?
Avi: Ummmm...