

Derandomization

Satyadev Nandakumar

October 14, 2026

Contents

1	Pseudorandom generators	2
2	Derandomization using PRGs	3
3	The Nisan-Wigderson Pseudorandom generator	3
3.1	Combinatorial Designs	3
3.2	Construction of the NW PRG	4

We now come to the main application of the techniques we have learned so far, namely, to derandomize algorithms. This is the process of removing the reliance on random coins of a randomized algorithm to yield a deterministic, efficient algorithm. Ideally, this process should be the same no matter the algorithm.

Why don't we just design a deterministic algorithm in the first place, rather than designing a randomized algorithm, and then derandomizing it? This is because for many problems, we still do not have any idea of designing an efficient deterministic algorithm, but very simple randomized algorithms exist.

A classic open problem currently known only to have efficient randomized algorithms but not deterministic efficient algorithms is polynomial identity testing: given two multivariate polynomials P_1 and P_2 , not explicitly, as a list of coefficients for example, but concisely as arithmetic circuits, determine whether they are identical. There is an efficient randomized algorithm using the Schwarz-Zippel lemma (originally discovered by Gauss), which considers $P_1 - P_2$, and evaluates it at many points - if there is any non-zero value, then we know that $P_1 - P_2$ is not identically zero. If all values evaluate to 0, then with high probability, $P_1 = P_2$. We can show, without much difficulty, that this algorithm errs only on one side - calling non-identical polynomials as identical - and the error probability is very small. We do not yet know an efficient deterministic algorithm.

(Stronger versions of the problem involve what are called *hitting sets* - this is a short list of vectors in a large enough finite fields such that for any non-zero polynomial P , there is some vector x in this list on which $P(x) \neq 0$.)

It will be nice if we have a uniform way to derandomize algorithms so that algorithm design can be made simple - come up with a randomized algorithm (which is usually a simple algorithm),

and then derandomize it using some standard technique, if available.

In what follows, we see that the hardness amplification techniques we studied so far yields a *pseudorandom generator* (abbreviated as PRGs), which can then be used to derandomize algorithms.

Note, before we begin, that no PRG is known to exist unconditionally. We construct a PRG under the assumption that E does not have small circuits - that is, there are problems which can be solved in linear exponential time, but cannot be solved by any polynomial-sized family of circuits. This is widely believed to be true, but we do not have a proof.

(We already know that E cannot be solved by polynomial-time Turing machines by the time hierarchy theorem, but we do not know the above result about circuits. If our assumption for PRGs had been that E is not P , then we already would have had unconditional PRGs, so we would know how to derandomize every BPP algorithm, yielding $P = BPP$. So the distinction between Turing machines and circuits is very important here.)

Converse. There is also the converse question: is it hard to derandomize BPP? Do we need circuit lower bounds to derandomize it? Is there some easier way? We see towards the end of this section that derandomizing BPP implies certain circuit lower bounds. Lower bounds for circuits are hard to obtain. Hence a full derandomization of BPP cannot be easy. Overall, we can say that the question of proving $BPP=P$ is essentially the same as showing circuit lower bounds, since these two problems imply each other.

1 Pseudorandom generators

A *pseudorandom distribution* is one which *fools* every small circuit - *i.e.* its acceptance probability does not vary much when random input (or random coins) are replaced by input sampled from this pseudorandom distribution.

Definition 1.1 (Pseudorandom distribution). Let $S \in \mathbb{N}$ denote the size of a circuit, and $\epsilon > 0$ be arbitrary. A probability distribution R over $\{0,1\}^m$ is an (S,ϵ) -*pseudorandom distribution*, if for every circuit C of size at most S ,

$$|\Pr[C(R) = 1] - \Pr[C(U_m) = 1]| \leq \epsilon, \quad (1)$$

where U_m denotes the uniform distribution over $\{0,1\}^m$.

Clearly, U_m itself is pseudorandom by definition. To generate an output according to U_m , we need m random bits. However, our need will be to construct pseudorandom distributions which do not require as many random bits. To get this, we use *pseudorandom generators*.

Definition 1.2 (Pseudorandom generators). Let $\text{Stretch} : \mathbb{N} \rightarrow \mathbb{N}$ be a function. A 2^n -time computable function $G : \{0,1\}^* \rightarrow \{0,1\}^*$ is an $\text{Stretch}(\ell)$ *pseudorandom generator* if for every string z , $|G(z)| = \text{Stretch}(|z|)$ and for every $\ell \in \mathbb{N}$, the distribution $G(U_\ell)$ is $(\text{Stretch}(\ell^3), 1/10)$ -pseudorandom.

In other words, G “stretches” any string z to a pseudorandom string with length $\text{Stretch}(|z|)$. The pseudorandom distribution generated from U_ℓ - random strings of length ℓ stretched to pseudorandom strings of length $\text{Stretch}(\ell)$ - can fool circuits of size $\text{Stretch}(\ell^3)$ with $1/10$ bias. The Stretch function is usually monotone increasing.

2 Derandomization using PRGs

The following is a general lemma relating the stretch of pseudorandom generators to the kind of complexity class that we can derandomize using it.

Lemma 2.1. *Suppose there is an $S(\ell)$ PRG for a time-constructible non-decreasing $S : \mathbb{N} \rightarrow \mathbb{N}$. Then for every polynomial-time computable function $\ell : \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{BPTIME}(S(\ell(n))) \subseteq \mathbf{DTIME}(2^{c\ell(n)})$ for some constant c .*

We only specify the most important consequence.

Corollary 2.2. *If there is a $2^{\epsilon \ell}$ pseudorandom generator for some constant $\epsilon > 0$, then $\mathbf{BPP} = \mathbf{P}$.*

Proof. (Proof of Theorem) Suppose $L \in \mathbf{BPTIME}(S(\ell(n)))$. Let A be an algorithm that on input $x \in \{0, 1\}^n$ runs in time $cS(\ell(n))$ for some constant c such that

$$\Pr_{r \sim \{0, 1\}^n} [A(x, r) = L(x)] \geq 2/3. \quad (2)$$

We run the algorithm with inputs $G(z)$ over all $z \in \{0, 1\}^\ell$ instead of using random strings $r \sim \{0, 1\}^n$.

So we do the following. On input x , run the algorithm A over all $z \in \{0, 1\}^{\ell(n)}$, with the “pseudorandom” input $G(z)$ in place of random coins. If the majority of these simulations accept x , then we accept x . Otherwise, we reject x . Since the output of the pseudorandom distribution does not cause a significant bias in acceptance by definition, we have

$$\Pr_{z \sim \{0, 1\}^{\ell(n)}} [A(x, G(z)) = L(x)] \geq 2/3 - 1/10. \quad (3)$$

Hence, the majority output will be correct. This is a deterministic algorithm that decides L in $\mathbf{DTIME}(2^{c\ell(n)})$. $\square$

3 The Nisan-Wigderson Pseudorandom generator

Definition 3.1. Let $\mathcal{I} = \{I_1, \dots, I_m\}$ be a family of subsets of $\{1, \dots, \ell\}$, with $|I_j| = n$ for each $j = 1, \dots, m$. Let $f : \{0, 1\}^m \rightarrow \{0, 1\}$ be a Boolean function.

The $(\mathcal{I}, f)$ -NW generator is the function

$$\text{NW}[\mathcal{I}, f](z) = f(z_{I_1}) \circ \dots \circ f(z_{I_m}). \quad (4)$$

To make this a pseudorandom generator, we want $\mathcal{I}$ to be a nice family of nearly disjoint sets called a *combinatorial design*, and f to be a function which is hard on average.

3.1 Combinatorial Designs

Definition 3.2. Let $d < n < \ell$. An (ℓ, n, d) -design is a family $\mathcal{I} = \{I_1, \dots, I_m\}$ of subsets of $\{1, \dots, \ell\}$ such that every set in the family has exactly n elements, and any two distinct sets in the family have at most d common elements.

We now show that there is a design that has exponential in d number of sets in the family. Hence, using a design as the family $\mathcal{I}$ in the NW-generator will lead to a PRG with exponential stretch. The proof is a deterministic, greedy construction. However, in the argument that the proof is correct, we will use a probabilistic argument.

Theorem 3.3. *There is an algorithm A that on input ℓ, n, d with $d < n$ and $10n^2/d < \ell$, outputs an (ℓ, n, d) -design in time $2^{O(\ell)}$.*

Proof. Start by setting the family of sets $\mathcal{I} = \emptyset$. We assume that the subsets of $\{1, \dots, \ell\}$ is linearly ordered according to some standard order. Inductively, select the next unchosen subset of $\{1, \dots, \ell\}$ such that it has exactly n elements, and has at most d common elements with any set already included in $\mathcal{I}$.

We now show that $\mathcal{I} = \{I_1, \dots, I_m\}$ has at least $2^{d/10}$ sets. Suppose not. It suffices to show that there is at least one more set I that can be added to the family $\mathcal{I}$ with the required properties.

We use the probabilistic method to show existence. Consider the following random experiment: Add each number $x \in \{0, \dots, \ell\}$ to a set I with probability $2n/\ell$ independently at random. Then the expected cardinality of I is $2n$. For any $I_j \in \mathcal{I}$, the expected value of $|I \cap I_j|$ is $2n^2/\ell < d/5$.

Using Chernoff bounds, it is possible to show that

$$P(|I| \geq \ell) \geq 0.9 \quad (5)$$

$$\forall j \in \{1, \dots, m\} P(|I \cap I_j| \geq d) \leq 0.52^{d/10}. \quad (6)$$

Since $m < 2^{d/10}$, with probability at least 0.4, both conditions will be satisfied. Hence a set I exists with the above two properties. We can remove extra elements in I to produce a new set with exactly n elements, with at most d elements in common with any set in $\mathcal{I}$. $\square$

3.2 Construction of the NW PRG

Using a combinatorial design and a hard-on-average function, the NW construction above yields a pseudorandom generator with exponential stretch. By the Corollary 2.2 we discussed above,

Theorem 3.4. *If $\mathcal{I}$ is an (ℓ, n, d) design with $2^{d/10}$ sets, and $f : \{0, 1\}^n \rightarrow \{0, 1\}$ has $H_{\text{avg}}(f) > 2^{2d}$, then the distribution $\text{NW}[\mathcal{I}, f](U_\ell)$ is $(H_{\text{avg}}(f)/10, 1/10)$ -pseudorandom.*

Proof. Let $S = H_{\text{avg}}(f)$. By assumption, $S > 2^{2d}$. Let us assume that $\text{NW}[\mathcal{I}, f](U_\ell)$ is **not** $(H_{\text{avg}}(f)/10, 1/10)$ -pseudorandom

By Yao's hybrid argument, there is an i , $1 \leq i \leq 2^{d/10}$ and an $\frac{S}{2}$ size circuit C such that

$$\Pr_{z \sim U_\ell, R = \text{NW}[\mathcal{I}, f](z)} [C(R_1, \dots, R_{i-1}) = R_i] \geq \frac{1}{2} + \frac{1}{102^{d/10}}. \quad (7)$$

By the definition of the NW generator, we know that $R_i = \text{NW}[\mathcal{I}, f](Z_{I_i})$ for every i . Hence, we have

$$\Pr_{z \sim U_\ell} [C(f(Z_{I_1}), \dots, f(Z_{I_{i-1}})) = f(Z_{I_i})] \geq \frac{1}{2} + \frac{1}{102^{d/10}}. \quad (8)$$

For this i , let Z_1 be the random variable corresponding to the variables of Z in I_i and Z_2 be the one corresponding to the variable of Z in $\{1, \dots, \ell\} \setminus I_i$. We can show that Z_1 and Z_2 are independent. Then we can write

$$\Pr_{Z \sim U_n, Z_2 \sim U_{\ell-n}} [C(f_1(Z_1, Z_2), \dots, f_{i-1}(Z_1, Z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}, \quad (9)$$

where every function f_j applies f to the coordinates of Z_1 corresponding to $I_j \cap I_i$ and Z_2 , to $I_j \setminus I_i$.

By the averaging argument, there is a single z_2 such that

$$\Pr_{Z \sim U_n} [C(f_1(Z_1, z_2), \dots, f_{i-1}(Z_1, z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}. \quad (10)$$

Since $\mathcal{I}$ is an (ℓ, n, d) -design, we have $|I_j \cap I_i| \leq d$ for all $j \neq i$. Hence the function $Z_1 \mapsto f_j(Z_1, z_2)$ depends on at most d coordinates of Z_1 . Hence f can be computed by a $d \cdot 2^d$ size circuit.

Hence, there is a circuit B of size $2^{d/10} \cdot d \cdot 2^d + S/2 < S$ such that

$$\Pr_{Z_1 \sim U_n} [B(Z_1) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}. \quad (11)$$

which contradicts the hardness of f . □