



8

Randomized Complexity

18 June 2025

Prof. Dr. Sebastian Wild

8 Randomized Complexity

- 8.1 Randomized Complexity Classes
- 8.2 Pseudorandom Generators
- 8.3 Excursion: Boolean Circuits
- 8.4 Derandomization
- 8.5 Nisan-Wigderson Pseudorandom Generator
- 8.6 Summary

The Power of Randomness

We've seen examples where randomized algorithms are provably more powerful ...
but how general are such improvements?

Before we consider algorithmic design techniques, we will consider the theoretical power of randomization:

Does randomization extend the range of problems solvable by polytime algorithms?

↪ back to *decision* problems.

8.1 Randomized Complexity Classes

Randomization for Decision Problems

- ▶ Recall: P and NP consider decision problems only

\rightsquigarrow equivalently: languages $L \subseteq \Sigma^*$

Can make some simplifications for algorithms:

- ▶ Only 3 sensible output values: 0, 1, $\boxed{?}$
- ▶ Unless specified otherwise, allow unlimited #random bits,
i.e., $random_A(x) = time_A(x)$ (Can't read more than one random bit per step)

Randomized Complexity Classes

Definition 8.1 (ZPP)

ZPP (*zero-error probabilistic polytime*) is the class of all languages L with a polytime **Las Vegas** algorithm A , i. e.,

- (a) $\exists c : \text{Time}_A(n) = O(n^c)$ as $n \rightarrow \infty$ (In particular: always terminate!)
- (b) $\mathbb{P}[A(x) = [x \in L]] \geq \frac{1}{2}$
- (c) $A(x) \neq [x \in L]$ implies $A(x) = \boxed{?}$

Definition 8.2 (BPP)

BPP (*bounded-error probabilistic polytime*) is the class of languages L with a polytime **bounded-error Monte Carlo** algorithm A , i. e.,

- (a) $\exists c : \text{Time}_A(n) = O(n^c)$ as $n \rightarrow \infty$
- (b) $\exists \varepsilon > 0 : \mathbb{P}[A(x) = [x \in L]] \geq \frac{1}{2} + \varepsilon$

Definition 8.3 (PP)

PP (*probabilistic polytime*) is the class of languages L with a polytime **unbounded-error Monte Carlo** algorithm: (a) as above (b) $\mathbb{P}[A(x) = [x \in L]] > \frac{1}{2}$.

Error Bounds

Remark 8.4 (Success Probability)

From the point of view of complexity classes, the success probability bounds are flexible:

- ▶ BPP only requires success probability $\frac{1}{2} + \varepsilon$, but using *Majority Voting*, we can also obtain any fixed success probability $\delta \in (\frac{1}{2}, 1)$.
- ▶ Similarly for ZPP, we can use probability amplification on Las Vegas algorithms

~> Unless otherwise stated,

for BPP and ZPP algorithms A , require $\mathbb{P}[A(x) = [x \in L]] \geq \frac{2}{3}$

But recall: this is *not* true for **unbounded** errors and class PP.

In fact, we have the following result:

Theorem 8.5 (PP can simulate nondeterminism)

$NP \cup \text{co-NP} \subseteq PP$.

~> Useful algorithms must avoid unbounded errors.

PP can simulate nondeterminism [2]

Proof (Theorem 8.5):



One-Sided Errors

In many cases, errors of MC algorithm are only *one-sided*.

Example: (simplistic) randomized algorithm for SAT:

Guess assignment, output $[\phi \text{ satisfied}]$.

(Note: This is not a MC algorithm, since we cannot give a fixed error bound!)

Observation: No false positives; unsatisfiable ϕ always yield 0.
... could this help?

Definition 8.6 (One-sided error Monte Carlo algorithms)

A randomized algorithm A for language L is a *one-sided-error Monte-Carlo (OSE-MC) algorithm* if we have

(a) $\mathbb{P}[A(x) = 1] \geq \frac{1}{2}$ for all $x \in L$, and

(b) $\mathbb{P}[A(x) = 0] = 1$ for all $x \notin L$.

↪ OSE-MC: $A(x) = 1$ must always be correct; $A(x) = 0$ may be a lie

One-Sided Error Classes

Definition 8.7 (RP, co-RP)

The classes RP and co-RP are the sets of all languages L with a polytime OSE-MC algorithm for L resp. \bar{L} .

Theorem 8.8 (Complementation feasible \rightarrow errors avoidable)

$\text{RP} \cap \text{co-RP} = \text{ZPP}$.

Proof:

See exercises.

Note the similarity to the wide open problem $\text{NP} \cap \text{co-NP} \stackrel{?}{=} \text{P}$.

For the latter, the common belief is $\text{NP} \cap \text{co-NP} \supsetneq \text{P}$, in sharp contrast to the randomized classes.

8.2 Pseudorandom Generators

Derandomization

- ▶ Suppose we have a **BPP** algorithm A , i. e., a polytime TSE-MC algorithm

$\rightsquigarrow Random_A(n)$ bounded

\rightsquigarrow There are at most $2^{Random_A(n)}$ different random-bit inputs ρ
and hence at most so many different computations for A on inputs $x \in \Sigma^n$

- ▶ The *derandomization* of A is a deterministic algorithm that simply simulates all these computations one after the other (and outputs the majority).
- ▶ In general, the exponential blowup makes this uninteresting.

▶ **But:** If $Random_A(n) \leq c \cdot \overset{= \log_2}{\lg}(n)$,
the derandomization of A runs in polytime: $n^c \cdot Time_A(n)$

⚡ Typical randomized algorithms use $\Omega(n)$, not $O(\log n)$ random bits.

Pseudorandom Generators

- ▶ “Typical randomized algorithms use $\Omega(n)$, not $O(\log n)$ random bits.”



But how would an algorithm actually *know* whether what we give it is truly random?

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```

<https://xkcd.com/221/>

- ▶ must somehow keep the random distribution . . .
in general not clear what “sufficiently random” would mean

↪ Breakthrough idea in TCS: *Pseudorandom Generators*

- ▶ generate an exponential number of bits from a n given truly random bits such that **no efficient** algorithm can distinguish them from truly random

↗ in a model to be specified

- ▶ **Key (Open!) Question:** *Do they exist?!*
- ▶ **Surprising answer:** We have good evidence in favor (!)

8.3 Excursion: Boolean Circuits

Boolean Circuits

For technical reasons (stay tuned ...), another model of computation more convenient than TM here.

Definition 8.9 (Boolean circuit)

An n -input *Boolean circuit* is a connected DAG $C = (V, E)$

- ▶ with n *sources* (labeled x_1, \dots, x_n)
- ▶ a single *sink* c (the output)
- ▶ any number of *gates* (non-sink vertices) labeled with \wedge, \vee , or \neg .
- ▶ All gates have in- and out-degree at most 2 (*fan-in* = *fan-out* = 2). (\neg is always unary)

The *value* of C , $C(x_1, \dots, x_n)$ for a given variable assignment is computed inductively: We assign the variable value to sources and apply the Boolean function at gates to inputs.

The *size* of C is the number of vertices $|C| = |V(C)|$.

A circuit C computes function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ if $\forall x \in \{0, 1\}^n : C(x) = f(x)$. ◀

Definition 8.10 (Circuit complexity)

The circuit complexity $\mathcal{H}(f)$ of a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is the size of the *smallest* Boolean circuit C that computes f . ◀

Formula vs. Circuit

Parity function: $P_n(x_1, \dots, x_n) = \bigoplus_{i=1}^n x_i = \sum_{i=1}^n x_i \bmod 2$ (odd number of 1-bits?)

► By associativity, $P_n(x_1, \dots, x_n) = P_{n-1}(x_1, \dots, x_{n-1}) \oplus x_n$

► also: $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$

\rightsquigarrow Can build a circuit for P_n using $5(n-1)$ gates

► Obvious boolean formula: (over basis $\{\wedge, \vee, \neg\}$)

$$P_n(x_1, \dots, x_n) = (x_n \wedge \neg P_{n-1}(x_1, \dots, x_{n-1})) \vee (\neg x_n \wedge P_{n-1}(x_1, \dots, x_{n-1}))$$

$\rightsquigarrow 5 \cdot 2^{n-1}$ operators

► optimal (assuming $n = 2^k$):

$$P_n(x_1, \dots, x_n) = (P_{n/2}(x_1, \dots, x_{n/2}) \cap \neg P_{n/2}(x_{n/2+1}, \dots, x_n)) \\ \vee (\neg P_{n/2}(x_1, \dots, x_{n/2}) \cap P_{n/2}(x_{n/2+1}, \dots, x_n))$$

$\rightsquigarrow \Theta(n^2)$ still much more than for circuits!

Circuit Complexity Classes

Poly-size circuits: (somewhat analogous to P , but not quite ...)

- ▶ P_{poly} = all functions computable by *polynomial-sized* circuits
- ▶ Can prove: $P \subseteq P_{\text{poly}}$

Theorem 8.11 (TM to circuit)

For $f \in \text{TIME}(T(n))$ and input size n , we can compute in polytime a circuit C for f on inputs of size n of size $|C| = O(T(n)^2)$. ◀

(Arora & Barak, Theorem 6.6)

- ▶ actually $P \subsetneq P_{\text{poly}}$:
circuits are *non-uniform* model of computation: *different circuit for each n*
↪ has some weird properties in general (P_{poly} contains a version of halting problem ...)
- ▶ Probably $NP \not\subseteq P_{\text{poly}}$ (unless polynomial hierarchy collapses)

allows some “cheating” that we use later

Circuit Lower Bounds:

- ▶ Can show: almost all Boolean functions f have *exponential* $\mathcal{C}(f)$ (counting argument)
- ▶ But: *Very* hard to prove circuit lower bounds for *concrete* functions f
 - ▶ Showing $\mathcal{H}(f)$ **exponential** for *any* $f \in NP$ would imply $P \neq NP$
 - ▶ Proven lower bounds on $\mathcal{H}(f)$ for explicit f are typically **linear** in n

Monte Carlo Circuits

We need a somewhat peculiar, weaker form of circuit complexity, where we assume that inputs $X \in \{0, 1\}^n$ are chosen *uniformly at random*.

Definition 8.12 (Average-case hardness)

The ρ -*average-case hardness* $\mathcal{H}_{avg}^\rho(f)$ of a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is the largest size S , such that every circuit C with $|C| \leq S$ we have $\mathbb{P}_X [C(X) = f(X)] < \rho$.
(Need circuits larger than $\mathcal{H}_{avg}^\rho(f)$ for confidence ρ .)

The *average-case hardness* of f then is $\mathcal{H}_{avg}(f) = \max \left\{ S : \mathcal{H}_{avg}^{\frac{1}{2} + \frac{1}{S}}(f) \geq S \right\}$.
(Allow larger circuits and worse confidence until f probabilistically computable)

Hypothesis 8.13 (Hard functions exist)

There exists a function $f \in \text{NP}$ with $\mathcal{H}_{avg}(f) = 2^{\Omega(n)}$.

!NOT PROVEN!

- **Deep result** (that we skip): From existence of function with large $\mathcal{H}(f)$, can conclude existence of function with large $\mathcal{H}_{avg}(f)$.

(see Arora & Barak Chapter 19)

- 3SAT probably has exponential $\mathcal{H}(f)$ (\approx ETH) (and other candidates exist)

Formalization Pseudorandom Generator

Definition 8.14 (Pseudorandom bits)

A r.v. $R \in \{0, 1\}^m$ is (S, ε) -*pseudorandom* if for every circuit C with $|C| \leq S$

$$\left| \mathbb{P}[C(R) = 1] - \mathbb{P}[C(\mathbf{U}_m) = 1] \right| < \varepsilon \quad \text{where} \quad \mathbf{U}_m \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^m)$$

*Pseudorandom bits are **indistinguishable** from truly random for any small circuit.*

think: fast-running algorithm

Definition 8.15 (Pseudorandom generator)

Let $S : \mathbb{N}_{\geq 1} \rightarrow \mathbb{N}_{\geq 1}$.

A function $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ computable in 2^n time ($G \in \text{TIME}(2^n)$) is an

$S(\ell)$ -*pseudorandom generator* ($S(\ell)$ -PRG) if

- (a) $|G(z)| = S(|z|)$ for every $z \in \{0, 1\}^*$
- (b) $\forall \ell \in \mathbb{N}_{\geq 1} : G(\mathbf{U}_\ell)$ is $(S(\ell)^3, \frac{1}{10})$ -pseudorandom.

Seeding a generator with ℓ truly random bits yields $S(\ell)$ pseudorandom bits.

8.4 Derandomization

Pseudorandom Generator for BPP Derandomization

The *Nisan-Wigderson construction* shows that the existence of any hard-on-average function implies a strong pseudorandom generator.

↖ exponentially many pseudorandom bits(!)

Theorem 8.16 (Strong NW PRG)

Assume Hypothesis 8.13, i. e., $f \in \text{TIME}(2^{O(n)})$ exists with $\mathcal{H}_{\text{avg}}(f) \geq S$ with $S(n) = 2^{\delta n}$ for a constant $\delta > 0$.

Then there is an $\varepsilon = \varepsilon(\delta)$ such that there is a $2^{\varepsilon \ell}$ -pseudorandom generator. ◀

(We will prove this over the course of the next subsection.)

BPP Derandomization

Theorem 8.17 (Hard-on-average function \rightarrow BPP = P)

Hypothesis 8.13 implies BPP = P.

Proof:

By Theorem 8.16, Hypothesis 8.13 implies a $S(\ell)$ -PRG $G : \{0, 1\}^\ell \rightarrow \{0, 1\}^{S(\ell)}$ with $S(\ell) = 2^{\varepsilon\ell}$.

Let $L \in \text{BPP}$. $\rightsquigarrow \exists$ algorithm A with $\text{Time}_A(n) \leq n^c$ (polytime) and $\mathbb{P}_R[A(x, R) = L(x)] \geq \frac{2}{3}$; here $R \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^m)$ for $m = \text{Random}_A(n) \leq \text{Time}_A(n) \leq n^c$.

We now obtain a **deterministic** polytime algorithm B as follows:

1. Replace R by $G(Z)$ for $Z \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^\ell)$ for $\ell = \ell(n) = \frac{c}{\varepsilon} \lg n$ so that $m \leq S(\ell) = 2^{\varepsilon\ell} = n^c$.
2. Instead of this probabilistic TM, simulate $A(x, G(z))$ for **all** possible $z \in \{0, 1\}^\ell$
3. Output the majority.

The trick here is that number of possible seeds z is $2^{\ell(n)} = n^c$, hence the running time remains polynomial and $B \in \text{P}$!

It remains to show that B accepts L .

(Intuition: A is too fast to notice a difference of more than $\frac{1}{10}$ between R and $G(Z)$.)

BPP Derandomization [2]

Proof (cont.):

Formally, assume towards a contradiction that there is an infinite sequence of x 's with $\mathbb{P}_Z[A(x, G(Z)) = L(x)] < \frac{2}{3} - \frac{1}{10} = 0.5\overline{6} > \frac{1}{2}$.

Then, we can build a *distinguisher* circuit C for the PRG: C simply computes the function $r \mapsto A(x, r)$, where x is hard-wired into the circuit C .

(Recall that $\mathbb{P}_R[A(x, R) = L(x)] \geq \frac{2}{3}$)

We don't have a circuit for A , just a TM;

but can convert A using Theorem 8.11 to a circuit C with $|C| = O((\text{Time}_A(n))^2) = O(n^{2c})$.

For sufficiently large n , $|C|$ is thus smaller than $S(\ell(n))^3 = n^{3c}$, so C is a valid distinguisher for the PRG. ⚡

Hence, the majority vote in B is correct

(for all but a finite number of inputs, which can be tested in constant time).

$\rightsquigarrow L \in \mathsf{P}$.

Consequences

- ~> Since the existence of hard-on-average functions is rather likely,
 - ▶ it must be assumed that randomization alone does **not** solve NP-hard problems;
 - ▶ ... and it seems that there is some heavy lifting going on in *Nisan-Wigderson*
- ~> Let's see what it does!

8.5 Nisan-Wigderson Pseudorandom Generator

Overview

- ▶ In this section, we will describe a conditional construction for pseudorandom generators based on the unproven hard-function hypothesis (Hypothesis 8.13).

*The higher the circuit lower bound $S(n)$ for our hard function f ,
the more pseudorandom bits we can generate from a fixed seed of ℓ truly random bits.*

- ▶ Key construction is due to Noam Nisan and Avi Wigderson (2023 Turing Award)
 - ▶ many further refinements followed
- ▶ *This is pretty cool stuff, but also complex. \rightsquigarrow Quantitative parts \notin exam.*

Theorem 8.18 (PRG from average-case hard function)

Let $S : \mathbb{N}_{\geq 1} \rightarrow \mathbb{N}_{\geq 1}$.

If there exists a function $f \in \text{TIME}(2^{O(n)})$ with $\mathcal{H}_{\text{avg}}(f)(n) \geq S(n)$ for all n ,
then there exists a $S(\delta\ell)^\delta$ -pseudorandom generator for some constant $\delta > 0$. ◀

This general result is for a refined construction and works also for weaker assumptions.

We will show the version sufficient for Theorem 8.16; see Arora & Barak Remark 20.8

Nisan-Wigderson Generator

The idea of the **Nisan-Wigderson (NW) generator** is to feed many (partially overlapping) subsets $I \in \mathcal{I}$ of ℓ truly random input bits into a (hard) function $f : \{0, 1\}^n \rightarrow \{0, 1\}$

$$\text{NW}_{\mathcal{I}}^f(Z) = f(Z_{I_1}) f(Z_{I_2}) \dots f(Z_{I_m})$$

where $Z \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^\ell)$ is the random seed and z_I for $I = \{i_1, \dots, i_n\}$ denotes $(z_{i_1}, \dots, z_{i_n})$

A key component is a sufficiently large subset system \mathcal{I} without too much overlap.

Definition 8.19 (Combinatorial Design)

For $\ell > n > d$, a family $\mathcal{I} = \{I_1, \dots, I_m\}$ of m subsets of $[\ell]$ is an (ℓ, n, d) -**design** if for all j and $k \neq j$,

- ▶ we have $|I_j| = n$ and
- ▶ $|I_j \cap I_k| \leq d$.

(We will eventually want to use this with $m = 2^{\varepsilon \ell}$.)

Probabilistic Method for Combinatorial Designs

Lemma 8.20 (NW Design)

There is an algorithm A that outputs on input (ℓ, n, d) with $\ell > n > d$ and $\ell > 10n^2/d$ an (ℓ, n, d) -design \mathcal{J} with $|\mathcal{J}| = 2^{d/10}$ subsets of $[\ell]$ in time $2^{O(\ell)}$.

Proof:

A is a simple greedy strategy: We start with $\mathcal{J} = \emptyset$. For $m \in [2^{d/10}]$, iterate over all 2^ℓ subsets of $[\ell]$ and include into \mathcal{J} the first set I with $\max_{J \in \mathcal{J}} |J \cap I| \leq d$.

To show: A succeeds. We use the probabilistic method!

Generate random I by picking each element $x \in [\ell]$ independently with probability $2n/\ell$.

By Chernoff:

$$(1) \mathbb{P}[|I| \geq n] \geq 0.9$$

$$(2) \mathbb{P}[|I \cap J| \geq d] \leq \frac{1}{2} \cdot 2^{-d/10} \text{ for any } J \in \mathcal{J}$$

Since $|\mathcal{J}| \leq 2^{d/10}$ and union bound on (2), $\mathbb{P}[\max_{J \in \mathcal{J}} |J \cap I| \geq d] \leq \frac{1}{2}$.

Hence, with probability at least $0.9 \cdot 0.5 = 0.45$, our random set I has intersection $\leq d$ with all old sets and $\geq n$ elements. Dropping elements until $|I| = n$ does not change that.

\rightsquigarrow In each step, we have probability ≥ 0.45 to succeed. So picking m random sets succeeds with probability $\geq 0.45^m > 0$, so some choice of sets \mathcal{J} as claimed must exist. ■

Unpredictable Next Bits

The second ingredient shows the (nontrivial) fact that having an unpredictable next bit implies pseudorandomness.

Definition 8.21 (unpredictable)

Let $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a function with $|G(x)| = S(|x|)$ for all $x \in \{0, 1\}^*$ (“stretch S ”).
 G is *unpredictable* if for every i and circuit C with $|C| \leq 2S(n)$ we have for $X \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^n)$ and $Y = G(X)$

$$\mathbb{P}_X[C(Y_1 \dots Y_{i-1}) = Y_i] \leq \frac{1}{2} + \frac{1}{10 \cdot S(\ell)}.$$



Unpredictable \rightarrow Pseudorandom

Theorem 8.22 (Yao's Theorem)

Let $S : \mathbb{N}_{\geq 1} \rightarrow \mathbb{N}_{\geq 1}$ be polytime computable and G as above with stretch S .

If G is unpredictable, then G is an $S(\ell)$ -pseudorandom generator.

Proof (Sketch):

Assume towards a contradiction that G is not a PRG.

Then there exists a circuit C that behaves substantively different on $G(\mathcal{U}(\{0,1\}^\ell))$ and $\mathcal{U}(\{0,1\}^{S(\ell)})$ bits: $\frac{1}{10}$ more or less likely to output 1.

For each i , we can construct a *predictor* circuit B_i from C :

Run C with $Y[1..i-1]$ followed by truly random bits $Z[i..S(\ell)]$;

if C outputs 1, output $Z[i]$, otherwise $1 - Z[i]$.

Note that B_0 executes C on purely random bits, $B_{S(\ell)}$ on purely pseudorandom bits.

C differs by $\frac{1}{10}$ on these, so (careful analysis shows that) we cannot have *all* $S(\ell)$ circuits B_i guess correctly only with prob. $\leq \frac{1}{2} + \frac{1}{10} \cdot \frac{1}{S(\ell)}$.

$\exists i$ such that B_i predicts Y_i correctly with prob. $\geq \frac{1}{2} + \frac{1}{10 \cdot S(\ell)}$,

so G is **not** unpredictable.

(For full details, see Arora & Barak, Theorem 20.10)

NW Pseudorandom Generator

Lemma 8.23 (NW Pseudorandom)

Let \mathcal{I} be an (ℓ, n, d) -design with $m = |\mathcal{I}| = 2^{d/10}$ and $f : \{0, 1\}^n \rightarrow \{0, 1\}$ a (hard) function with $\mathcal{H}_{avg}(f) > 2^{2d}$. Then $\text{NW}_{\mathcal{I}}^f(\mathcal{U}(\{0, 1\}^\ell))$ is $(\frac{1}{10}\mathcal{H}_{avg}(f), \frac{1}{10})$ -pseudorandom. ◀

Proof (Sketch):

By Yao's Theorem, we only need to show that NW is unpredictable; we will show that a predictor circuit C would lead to a small circuit B for f .

Let $S = \mathcal{H}_{avg}(f)$, i. e., on inputs of size n , f requires circuits larger than $S = S(n) > 2^{2d}$ to be computed with confidence $\geq \frac{1}{2} + \frac{1}{S}$.

Now assume towards a contradiction that NW is **not** unpredictable.

↪ $\exists i \in [m]$ and circuit C with $|C| \leq 2m < S/2$ and

$$\mathbb{P}_Z[C(R[1..i-1]) = R[i]] \geq \frac{1}{2} + \frac{1}{10m} \quad \text{where } R = \text{NW}(Z) \quad \text{and} \quad Z \stackrel{\mathcal{D}}{=} \mathcal{U}(\{0, 1\}^\ell) \quad (*)$$

Recall that $R[j] = f(Z_{I_j})$;

by renaming, assume $R[i] = f(Z[1..n])$ and write $Z_1 = Z[1..n]$ and $Z_2 = Z(n..\ell)$.

$$\rightsquigarrow \mathbb{P}_Z[C(f(Z_{I_1}) \dots f(Z_{I_{i-1}})) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10m}$$

NW Pseudorandom Generator [2]

Proof (cont):

Averaging Principle: For event $A = A(X, Y)$ holds $\exists x : \mathbb{P}_Y[A(x, Y)] \geq \mathbb{P}_{X,Y}[A(X, Y)]$

(effectively the probabilistic method on event probabilities)

We apply this to event $A(Z_2, Z_1) = \{C(f(Z_{I_1}) \dots f(Z_{I_{i-1}})) = f(Z_1)\}$ and Z_2 .

So there are $n - \ell$ bits z_2 , so that:

$$\rightsquigarrow \mathbb{P}_{Z_1}[C(f(Z_{I_1}) \dots f(Z_{I_{i-1}})) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10m} \quad \text{with} \quad Z = Z_1 z_2$$

Since \mathcal{I} is an (ℓ, n, d) -design, each $f(Z_{I_j})$ has $\leq d$ bits from Z_1 ; the other $n - d$ are hardcoded bits from z_2 . So we can compute $f(Z_{I_j})$ with a circuit of size $d2^d$ (CNF formula suffices).

Putting all $i - 1 \leq m = 2^{d/10}$ of these circuits and C together, we obtain a circuit B of size $2^{d/10} \cdot d2^d + S/2 < S$, with

$$\mathbb{P}_{Z_1}[B(Z_1) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}.$$

This contradicts the fact that $S = \mathcal{H}_{avg}(f)$. ■

Picking Parameters

► Generic algorithm:

► Setup: $f \in \text{TIME}(2^{O(n)})$ and $S : \mathbb{N}_{\geq 1} \rightarrow \mathbb{N}_{\geq 1}$ with $\mathcal{H}_{\text{avg}}(f) \geq S$

► Input: Random seed $Z \in \{0, 1\}^\ell$ (truly random bits)

► Algorithm: $n := \max \left\{ n : \frac{10n^2}{\lg S(n)/10} < \ell \right\}$

$$d := \lg S(n)/10$$

$\mathcal{J} := (\ell, n, d)$ -design with $m = |\mathcal{J}| = 2^{d/10}$ (algorithm from Lemma 8.20)

Output $\text{NW}_{\mathcal{J}}^f(Z)$

\rightsquigarrow By Lemma 8.23, the output is $(S(n)/10, \frac{1}{10})$ pseudorandom.

► Parameters for Theorem 8.16

► Assuming Hypothesis 8.13: f exists with $\mathcal{H}_{\text{avg}}(f) \geq S$ with $S(n) = 2^{\delta n}$.

► The inequality becomes $\ell > \frac{10n^2}{\lg S(n)/10} = \frac{100n^2}{\delta n} = \frac{100}{\delta}n$, so $n \approx \frac{\delta}{100}\ell$.

► $d = \lg S(n)/10 = \delta n/10 = \frac{\delta^2}{1000}\ell$

► NW can generate $m = 2^{d/10} = 2^{\delta n/100} = 2^{(\frac{\delta}{100})^2 \ell}$ pseudorandom bits

► Pseudorandom against circuits of size $S(n)/10 = 2^{\delta^2 \ell / 100} / 10 \underset{\ell \rightarrow \infty}{\gg} 2^{3(\frac{\delta}{100})^2 \ell} = m^3$

$\rightsquigarrow \text{NW}_{\mathcal{J}}^f$ is a $2^{\varepsilon \ell}$ -pseudorandom generator with $\varepsilon = (\delta/100)^2$

8.6 Summary

Overview Randomized Complexity Classes

Proven facts:

- ▶ $P \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP$
- ▶ $RP \subseteq NP$
- ▶ $NP \cup \text{co-NP} \subseteq PP$
- ▶ $ZPP = RP \cap \text{co-RP}$
- ▶ $NP \subseteq \text{co-RP} \implies NP = ZPP$

Widely held belief (but not proven):

- ▶ $P = BPP$
and hence $P = ZPP = RP = BPP$
- ▶ $BPP \subsetneq NP \subseteq PP$

Consequences

- ▶ **don't** try to solve NP-hard problems *exactly* using randomization in polytime
- ▶ **do** seek *easier and faster* algorithms for problems in **P!**
They often exist!
- ▶ **do** seek randomized algorithms for problems of unknown complexity status
Some exist!