8 Randomized Complexity

18 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-23 16:30 H`

Outline

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?

$\rightsquigarrow$ back to *decision* problems.

8.1 Randomized Complexity Classes

Randomization for Decision Problems

- Recall: P and NP consider decision problems only

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

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) $\exists c \;:\; Time_A(n) \;=\; O(n^c)$ as $n \to \infty$ (In particular: always terminate!)

(b) $\mathbb{P}\big[A(x) = [x \in L]\big] \;\geq\; \frac{1}{2}$

(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) $\exists c \;:\; Time_A(n) \;=\; O(n^c)$ as $n \to \infty$

(b) $\exists \varepsilon > 0 \;:\; \mathbb{P}\big[A(x) = [x \in L]\big] \;\geq\; \frac{1}{2} + \varepsilon$ ◀

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,

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

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)

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

⇝ Useful algorithms must avoid unbounded errors.

PP can simulate nondeterminism [1]

Proof (Theorem 8.5):

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 $x \notin L$. ◀

$\rightsquigarrow$ $\boxed{\text{OSE-MC: } A(x) = 1 \text{ must always be correct; } A(x) = 0 \text{ may be a lie}}$

One-Sided Error Classes

Definition 8.7 (RP, co-RP)

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

Theorem 8.8 (Complementation feasible → errors avoidable)

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

Proof:
See exercises. ■

Note the similarity to the wide open problem $\mathsf{NP} \cap \mathsf{co\text{-}NP} \stackrel{?}{=} \mathsf{P}$.
For the latter, the common belief is $\mathsf{NP} \cap \mathsf{co\text{-}NP} \supsetneq \mathsf{P}$, 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.

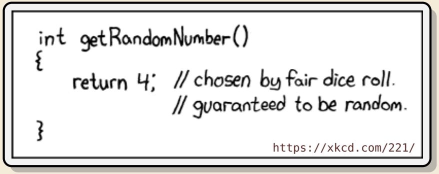
- **But:** If $Random_A(n) \le c \cdot \underset{\uparrow}{\overset{= \log_2}{\lg}}(n)$,
the derandomization of A runs in polytime: $n^c \cdot Time_A(n)$

ϟ 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?



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

$\rightsquigarrow$ 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 (!)

Excursion: Boolean Circuits Complexity

Definition 8.9 (Boolean circuit)

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

- with n *sources* (labeled $x_1, \ldots, x_n$)
- 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). (¬ is always unary)

The *value* of C, $C(x_1, \ldots, x_n)$ 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 $f : \{0,1\}^n \to \{0,1\}$ if $\quad \forall \boldsymbol{x} \in \{0,1\}^n : C(\boldsymbol{x}) = f(\boldsymbol{x})$.

Definition 8.10 (Circuit complexity)

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

Excursion: Formula vs. Circuit

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

XOR

- By associativity, $P_n(x_1,\ldots,x_n) = P_{n-1}(x_1,\ldots,x_{n-1}) \oplus x_n$
- also: $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$

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

- Obvious boolean formula: (over basis $\{\wedge, \vee, \neg\}$)
  $P_n(x_1,\ldots,x_n) = \big(x_n \wedge \neg P_{n-1}(x_1,\ldots,x_{n-1})\big) \vee \big(\neg x_n \wedge P_{n-1}(x_1,\ldots,x_{n-1})\big)$

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

- optimal (assuming $n = 2^k$):

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

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

Excursion: Circuits Complexity Classes

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

- $\mathsf{P}_{/\mathsf{poly}}$ = all functions computable by *polynomial-**sized*** circuits
- Can prove: $\mathsf{P} \subseteq \mathsf{P}_{/\mathsf{poly}}$

  Theorem 8.11 (TM to circuit)
  For $f \in TIME(T(n))$ 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 $\mathsf{P} \subsetneq \mathsf{P}_{/\mathsf{poly}}$:
    allows some "cheating" that we use later
    circuits are *non-uniform* model of computation: different circuit for each n

  $\rightsquigarrow$ has some weird properties in general ($\mathsf{P}_{/\mathsf{poly}}$ contains a version of halting problem . . .)
- Probably $\mathsf{NP} \not\subseteq \mathsf{P}_{/\mathsf{poly}}$ (unless polynomial hierarchy collapses)

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* $f \in \mathsf{NP}$ would imply $\mathsf{P} \neq \mathsf{NP}$

Excursion: 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* $\mathcal{H}^{\rho}_{avg}(f)$ of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the largest size S, such that every circuit C with $|C| \le S$ we have $\mathbb{P}\big[C(X) = f(X)\big] < \rho$.

(Need circuits larger than $\mathcal{H}^{\rho}_{avg}(f)$ for confidence ρ.)

The *average-case hardness* then is $\mathcal{H}_{avg}(f) = \max\Big\{S : \mathcal{H}^{\frac{1}{2}+\frac{1}{S}}_{avg} \ge S\Big\}$.

(Allow larger circuits and worse confidence until f probabilistically computable)

Hypothesis 8.13 (Hard functions exist)

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

- **Deep result** (that we skip): From existence of function with large $\mathcal{H}(f)$, can conclude existence of function with large $\mathcal{H}_{\boldsymbol{avg}}(f)$.
  (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. $\boldsymbol{R} \in \{0,1\}^m$ is (S, ε)-*pseudorandom* if for every circuit C with $|C| \le S$

$$\Big|\mathbb{P}\big[C(\boldsymbol{R}) = 1\big] - \mathbb{P}\big[C(\boldsymbol{U}_m)\big]\Big| < \varepsilon \qquad \text{where} \qquad \boldsymbol{U}_m \overset{\mathcal{D}}{=} \mathcal{U}(\{0,1\}^m)$$ ◀

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

think: fast-running algorithm

Definition 8.15 (Pseudorandom generator)

Let $S : \mathbb{N}_{\ge 1} \to \mathbb{N}_{\ge 1}$.
A function $G : \{0,1\}^\star \to \{0,1\}^\star$ computable in 2^n time ($G \in \mathit{TIME}(2^n)$) is an $S(\ell)$-*pseudorandom generator* ($S(\ell)$-*PRG*) if

(a) $|G(z)| = S(|z|)$ for every $z \in \{0,1\}^\star$

(b) $\forall \ell \in \mathbb{N}_{\ge 1} \; : \; G(\boldsymbol{U}_\ell)$ is $\big(S(\ell)^3, \frac{1}{10}\big)$-pseudorandom. ◀

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

8.3 Derandomization of BPP?

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., $f \in TIME(2^{O(n)})$ exists with $\mathcal{H}_{avg}(f) \geq S$ 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 → $\mathsf{BPP} = \mathsf{P}$)

Hypothesis 8.13 implies $\mathsf{BPP} = \mathsf{P}$.

Proof:

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

Let $L \in \mathsf{BPP}$. $\rightsquigarrow$ $\exists$ algorithm A with $Time_A(n) \le n^c$ (polytime) and $\mathbb{P}_R[A(x,R) = L(x)] \ge \frac{2}{3}$; here $R \overset{\mathcal{D}}{=} \mathcal{U}(\{0,1\}^m)$ for $m = Random_A(n) \le Time_A(n) \le n^c$.

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

1. Replace R by $G(Z)$ for $Z \overset{\mathcal{D}}{=} \mathcal{U}(\{0,1\}^\ell)$ for $\ell = \ell(n) = \frac{c}{\varepsilon} \lg n$ so that $m \le S(\ell) = 2^{\varepsilon\ell} = n^c$.
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 $B \in \mathsf{P}$!

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

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; we can convert A using Theorem 8.11 to a circuit C with $|C| = O\left((Time_A(n))^2\right) = O(n^{2c})$.

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

Hence, the majority vote in B is correct (for all but a finite number of inputs, which can be tested in constant time). $\rightsquigarrow$ $L \in P$. ■

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 this *Nisan-Wigderson construction!*
  - ⇝ Let's see what's inside it!

8.4 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 $S : \mathbb{N}_{\ge 1} \to \mathbb{N}_{\ge 1}$.
If there exists a function $f \in \mathit{TIME}(2^{O(n)})$ with $\mathcal{H}_{avg}(f)(n) \ge S(n)$ 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 $I \in \mathcal{J}$ of ℓ truly random input bits into a (hard) function $f : \{0,1\}^n \to \{0,1\}$

$$\mathrm{NW}^f_{\mathcal{J}}(\mathrm{Z}) \quad = \quad f(\mathrm{Z}_{I_1})\, f(\mathrm{Z}_{I_2}) \ldots f(\mathrm{Z}_{I_m})$$

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

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

Definition 8.19 (Combinatorial Design)

For $\ell > n > d$, a family $\mathcal{J} = \{I_1, \ldots, I_m\}$ 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 $\mathcal{I}$ with $|\mathcal{I}| = 2^{d/10}$ subsets of $[\ell]$ in time $2^{O(\ell)}$.

Proof:

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

To show: A succeeds. We use the probabilistic method! If we create 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) $\mathbb{P}[|I \cap J| \geq d] \ \leq \ \frac{1}{2} \cdot 2^{-d/10}$ for any $J \in \mathcal{I}$

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

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 random elements until $|I| = n$ does not change that.

Hence, 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 $\mathcal{I}$ 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 $G : \{0,1\}^\star \to \{0,1\}^\star$ be a polytime-computable function with $|G(x)| = \ell(|x|)$ for all $x \in \{0,1\}^\star$ (*stretch* ℓ).
G is *unpredictable* if for every polytime PTM B,
we have for $X \overset{\mathcal{D}}{=} \mathcal{U}(\{0,1\}^n)$, $Y = G(X)$ and $I \overset{\mathcal{D}}{=} [1..\ell(n)]$ and all c that

$$\mathbb{P}_{X,I}\Big[B(\underset{\substack{\uparrow \\ \text{give } B \text{ time } n^{O(1)}}}{1^n}\, Y_1 \ldots Y_{I-1}) = Y_I\Big] \;\le\; \frac{1}{2} + o(n^{-c})$$

(We require B to predict a randomly chosen bit I, so it must work for all positions.)

Circuit version: G is *unpredictable* if for every i and circuit C with $|C| \le 2\ell(n)$ we have

$$\mathbb{P}_{X,I}\big[C(Y_1 \ldots Y_{i-1}) = Y_i\big] \;\le\; \frac{1}{2} + o(n^{-c}).$$

Unpredictable → Pseudorandom

Theorem 8.22 (Yao's Theorem)

Let $\ell : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ be polytime computable and G as above with stretch ℓ.
If G is unpredictable, then G is an $\ell(n)$-pseudorandom generator.

Proof (Sketch):

If G was not a PRG, there is an algorithm A that behaves substantively different on $G(\mathcal{U}(\{0,1\}^n))$ and $\mathcal{U}(\{0,1\})$ bits, ε more likely to output 1.
We construct PTM B from A:
Run A with $Y[1..I-1]$ followed by truly random bits $Z[I..\ell(n)]$; if A outputs 1 output $Z[I]$, otherwise $1 - Z[I]$.
Careful analysis shows that B predicts Y_I correctly with prob. $\geq \frac{1}{2} + \varepsilon/\ell(n)$, so G is not unpredictable.
The proof can be adapted to the circuit version, too.

(For full details, see Arora & Barak, Theorem 9.11) ∎

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 $f : \{0,1\}^n \to \{0,1\}$ a (hard) function with $\mathcal{H}_{avg}(f) > 2^{2d}$. Then $\mathrm{NW}_{\mathcal{I}}^{f}(\mathcal{U}(\{0,1\}^{\ell}))$ 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 $\geq \frac{1}{2} + \frac{1}{S}$.

Towards **refuting** the circuit-predictability of NW, suppose for some $i \in [m]$ there is a circuit C with $|C| \leq m/2 < S/2$ and

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

Recall that $R[j] = f(Z_{I_j})$; by renaming, let $R[i] = f(Z[1..n])$ and write $Z_1 = Z[1..n]$ and $Z_2 = Z(n..\ell]$.

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

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 $C\big(f(Z_{I_1}) \dots f(Z_{I_{i-1}})\big) = f(Z_1)$ and Z_2. So there are $n - \ell$ bits z_2, so that:

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

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

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

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

Picking Parameters

- **Generic algorithm:**
  - Setup: $f \in \mathit{TIME}(2^{O(n)})$ and $S : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ with $\mathcal{H}_{avg}(f) \geq S$
  - Input: Random seed $Z \in \{0,1\}^\ell$ (truly random bits)
  - Algorithm: $n := \max\left\{n : \dfrac{10n^2}{\lg S(n)/10} < \ell\right\}$

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

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

    Output $\mathrm{NW}^f_{\mathcal{I}}(\mathrm{Z})$

  $\rightsquigarrow$ 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 $\mathcal{H}_{avg}(f) \geq S$ 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 $S(n)/10 = 2^{\delta^2 \ell/100}/10 \underset{\ell \to \infty}{\gg} 2^{3(\frac{\delta}{100})^2 \ell} = m^3$

  $\rightsquigarrow$ $\mathrm{NW}^f_{\mathcal{I}}$ is a $2^{\varepsilon \ell}$-pseudorandom generator with $\varepsilon = (\delta/100)^2$

8.5 Summary

Overview Randomized Complexity Classes

Proven facts:

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

Widely held belief (but not proven):

- $\mathsf{P} = \mathsf{BPP}$
  and hence $\mathsf{P} = \mathsf{ZPP} = \mathsf{RP} = \mathsf{BPP}$
- $\mathsf{BPP} \subsetneq \mathsf{NP} \subseteq \mathsf{PP}$

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!