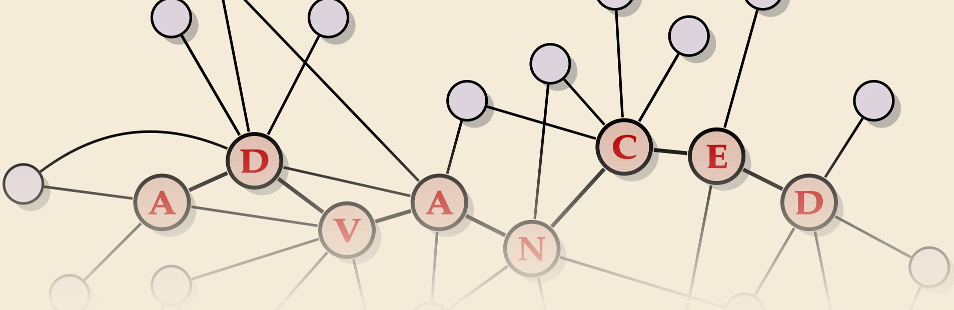


8 Randomized Complexity

18 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-24 09:00`

Outline

8 Randomized Complexity

The Power of Randomness

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

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?

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 $L \subseteq \Sigma^\star$

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)

$\leq$ always

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 $A(x) = \boxed{?}$

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

$\forall x \in \Sigma^*$

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

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

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

$\rightsquigarrow$ Unless otherwise stated,

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

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,

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

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 ∪ co-NP ⊆ PP.

⇝ Useful algorithms must avoid unbounded errors.

PP can simulate nondeterminism [1]

Proof (Theorem 8.5):

PP always allows polytime preprocessing

Given any $L \in NP$, we can use reduction $L \leq_p SAT$ (NP-complete)

$\leadsto$ suffices to show $SAT \in PP$

(TAUT is co-NP-complete $\leadsto$ works similarly for co-NP $\subseteq$ PP)

Given unbounded error MC algo A for SAT (polytime)

Given φ of length n over k variables

$A(\varphi)$: (1) Generate a (uniformly) random assignment $V: \{x_1, \ldots, x_k\} \to \{0,1\}$ (k random bits) $O(k)$

(2) If $V(\varphi) = 1$, output 1 $O(n)$

(3) Otherwise output $\mathcal{B}(p)$ $p = \frac{1}{2} - \frac{1}{2^{k+1}} < \frac{1}{2}$ $O(k)$

PP can simulate nondeterminism [2]

Proof (Theorem 8.5): running time polytime ✓

correctness : $\mathbb{P}[A(\varphi) = [\varphi \text{ sat.}]] \overset{?}{>} \frac{1}{2}$

- $\varphi \in SAT$ $\exists$ sat. assignment for $(x_1, \ldots, x_k)$

$$\mathbb{P}[\text{ step (2) succeeds}] \geq \frac{1}{2^k}$$

independence

$$\mathbb{P}[A(\varphi) = 0] = \mathbb{P}[V(\varphi) = 0] \cdot \mathbb{P}[B(p) = 0]$$

$$\leq \left(1 - \frac{1}{2^k}\right) \cdot \left(\frac{1}{2} + \frac{1}{2^{k+1}}\right) < \frac{1}{2}$$

- $\varphi \notin SAT$ $\mathbb{P}[V(\varphi) = 1] = 0$

$$\mathbb{P}[A(\varphi) = 1] = 1 \cdot \mathbb{P}[B(p) = 1] = p < \frac{1}{2}$$

$\Rightarrow \mathbb{P}[A(\varphi) = [\varphi \text{ sat.}]] > \frac{1}{2}.$

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?

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?

others: TSE-MC

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

One-Sided Error Classes

Definition 8.7 (RP, co-RP)

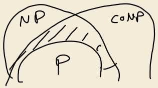
The classes RP and $\mathsf{co\text{-}RP}$ are the sets of all languages L with a polytime OSE-MC algorithm for L resp. $\overline{L}$. ◀

Theorem 8.8 (Complementation feasible → errors avoidable)

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

Proof:

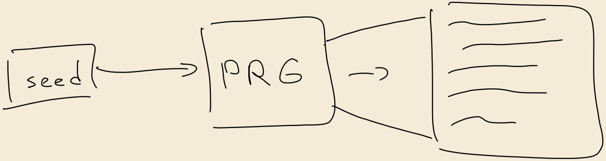
See exercises. ■



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

conventional definition (programming)



pseudorandom numbers = deterministic function of seed

good PRG = output passes statistical tests for randomness

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 $x \in \Sigma^n$

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

$2^{c \lg n} = (2^{\lg n})^c = n^c$

Derandomization

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

⇝ $Random_A(n)$ bounded

⇝ 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 $Random_A(n) \le c \cdot \overset{= \log_2}{\lg}(n)$,
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.

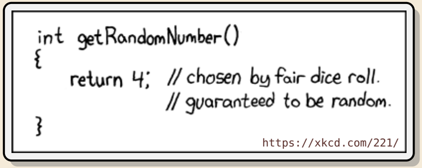
Pseudorandom Generators

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

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?

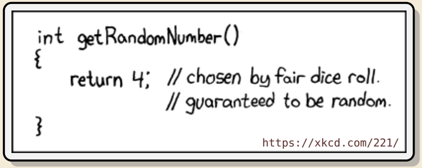


https://xkcd.com/221/

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

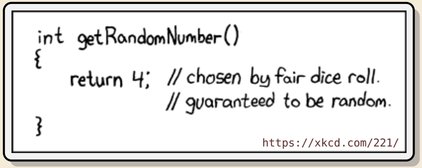
    in a model to be specified

  - **Key (Open!) Question:** *Do they exist?!*

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 (!)

8.3 Excursion: Boolean Circuits

Boolean Circuits

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

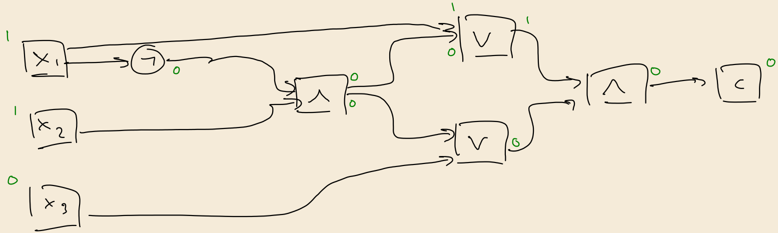
Boolean Circuits

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 $x_1, \ldots, x_n$)
- 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)



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 $x_1, \ldots, x_n$)
- 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.

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 $x_1, \ldots, x_n$)
- 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). ($\neg$ 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 $|C| = |V(C)|$.

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 $x_1, \ldots, x_n$)
- 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). ($\neg$ 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 $\forall \boldsymbol{x} \in \{0,1\}^n : C(\boldsymbol{x}) = f(\boldsymbol{x})$. ◀

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 $x_1, \ldots, x_n$)
- 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). ($\neg$ 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 $\forall \boldsymbol{x} \in \{0,1\}^n : C(\boldsymbol{x}) = f(\boldsymbol{x})$. ◀

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. ◀

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

Formula vs. Circuit

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

- 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

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

$(x_1 \oplus x_2 \oplus x_3) \oplus (x_4 \oplus x_5 \oplus x_6)$

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

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

⇝ $5 \cdot 2^{n-1}$ operators

- optimal (assuming $n = 2^k$):
  $$P_n(x_1, \ldots, x_n) = (P_{n/2}(x_1, \ldots, x_{n/2}) \cap \neg P_{n/2}(x_{n/2+1}, \ldots, x_n)) \vee (\neg P_{n/2}(x_1, \ldots, x_{n/2}) \cap P_{n/2}(x_{n/2+1}, \ldots, x_n))$$

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

Circuit Complexity Classes

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

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

$\forall n\ \exists C_n$: C_n computes $f|_{\{0,1\}^n}$
and $|C_n| = O(n^d)$

TM can always simulate circuit for fixed n

Circuit 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)

time in TM $\hat{=}$ size of circuit

Circuit 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 . . .)

any $L \subseteq \{1^n : n \in \mathbb{N}\} \in \mathsf{P}_{/\mathsf{poly}}$

Circuit 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 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 $f \in \mathit{TIME}(T(n))$ 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*
  - ⇝ 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* $\mathcal{C}(f)$ $\not\in$ NP (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 $\mathsf{P} \neq \mathsf{NP}$
  - 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 $\boldsymbol{X} \in \{0,1\}^n$ 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 $\mathbb{P}\left[C(\boldsymbol{X}) = f(\boldsymbol{X})\right] < \rho$.

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

Monte Carlo Circuits

We need a somewhat peculiar, weaker form of circuit complexity, where we assume that inputs $\boldsymbol{X} \in \{0,1\}^n$ 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 $|C| \leq S$ we have $\mathbb{P}\big[C(\boldsymbol{X}) = f(\boldsymbol{X})\big] < \rho$.
(Need circuits larger than $\mathcal{H}^{\rho}_{avg}(f)$ for confidence ρ.)

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

Monte Carlo Circuits

We need a somewhat peculiar, weaker form of circuit complexity, where we assume that inputs $\boldsymbol{X} \in \{0,1\}^n$ 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 $|C| \leq S$ we have $\mathbb{P}\big[C(\boldsymbol{X}) = f(\boldsymbol{X})\big] < \rho$.

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

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

(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 $\mathcal{H}_{avg}(f) = 2^{\Omega(n)}$.

!NOT PROVEN!

Monte Carlo Circuits

We need a somewhat peculiar, weaker form of circuit complexity, where we assume that inputs $\boldsymbol{X} \in \{0,1\}^n$ 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 $f : \{0,1\}^n \to \{0,1\}$ is the largest size S, such that every circuit C with $|C| \le S$ we have $\mathbb{P}\big[C(\boldsymbol{X}) = f(\boldsymbol{X})\big] < \rho$.

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

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

(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 $\mathcal{H}_{avg}(f) = 2^{\Omega(n)}$. **!NOT PROVEN!**

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

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

◀

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

think: fast-running algorithm

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 $|C| \leq S$

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

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

think: fast-running algorithm

Definition 8.15 (Pseudorandom generator)

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

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

(b) $\forall \ell \in \mathbb{N}_{\geq 1} \ : \ G(\boldsymbol{U}_\ell)$ is $\left(S(\ell)^3, \frac{1}{10}\right)$-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., $f \in TIME(2^{O(n)})$ 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 → BPP = P)

Hypothesis 8.13 implies BPP = P.

BPP Derandomization

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

Hypothesis 8.13 implies BPP = P.

Proof:

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

BPP Derandomization

Theorem 8.17 (Hard-on-average function → BPP = 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 $L \in \mathsf{BPP}$.

BPP Derandomization

Theorem 8.17 (Hard-on-average function → BPP = 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 $\overset{\text{PTM}}{}$ 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 $m = Random_A(n) \le Time_A(n) \le n^c$.

BPP Derandomization

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

Hypothesis 8.13 implies BPP = P.

Proof:

By Theorem 8.16, Hypothesis 8.13 implies a $S(\ell)$-PRG $G : \{0,1\}^{\ell} \to \{0,1\}^{S(\ell)}$ 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 $B \in \mathsf{P}$!

BPP Derandomization

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

Hypothesis 8.13 implies BPP = P. ◀

Proof:

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

Let $L \in$ 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 $B \in$ 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.): $B(x) \neq L(x) \rightsquigarrow$ majority vote wrong

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

$< \frac{1}{2} <$

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

$\varepsilon = \frac{1}{10}$

Definition 8.14 (Pseudorandom bits)

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

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

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

think: fast-running algorithm

Definition 8.15 (Pseudorandom generator)

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

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

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

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

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 $|C| = O\big((Time_A(n))^2\big) = O(n^{2c})$.

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 $|C| = O\big((Time_A(n))^2\big) = O(n^{2c})$.

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

Definition 8.14 (Pseudorandom bits)
A r.v. $\boldsymbol{R} \in \{0,1\}^m$ is (S,ε)-*pseudorandom* if for every circuit C with $|C| \leq S$

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

Pseudorandom bits are ***indistinguishable*** *from truly random* ***for any small circuit****.*
think: fast-running algorithm

Definition 8.15 (Pseudorandom generator)
Let $S : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$.
A function $G : \{0,1\}^\star \to \{0,1\}^\star$ computable in 2^n time ($G \in TIME(2^n)$) is an $S(\ell)$-*pseudorandom generator* ($S(\ell)$-*PRG*) if

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

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

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

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 $|C| = O\big((Time_A(n))^2\big) = O(n^{2c})$.

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).
$\rightsquigarrow \quad L \in P$. ∎

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 *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

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.*

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 $S : \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$.
If there exists a function $f \in TIME(2^{O(n)})$ with $\mathcal{H}_{avg}(f)(n) \geq S(n)$ for all n,
then there exists a $S(\delta\ell)^{\delta}$-pseudorandom generator for some constant $\delta > 0$.

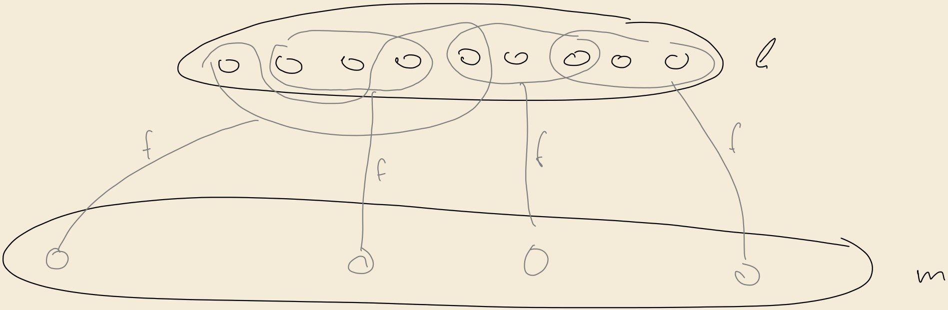
This general result is for a refined construction and works also for weaker assumptions.
We will show the version sufficient for Theorem 8.16; see Arora & Barak Remark 20.8

Nisan-Wigderson Generator

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

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

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 $(z_{i_1}, \ldots, z_{i_n})$



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 $f : \{0,1\}^n \to \{0,1\}$

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

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

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 $\mathcal{I} = \{I_1, \ldots, I_m\}$ 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 $\underbrace{|\mathcal{I}| = 2^{d/10}}_{m}$ subsets of $[\ell]$ in time $2^{O(\ell)}$.

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 $\max_{J \in \mathcal{J}} |J \cap I| \le d$.

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.

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!

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

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

By Chernoff:

$\mathbb{E}[I] = 2n$

(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 $J \in \mathcal{I}$

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 $2n/\ell$.

"bad"

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 $J \in \mathcal{J}$

$$\bigcup_{J\in\mathcal{J}} \mathbb{P}[|I\cap J| \geq d] \leq \underbrace{|\mathcal{J}|}_{\leq 2^{d/10}} \cdot \frac{1}{2} 2^{-d/10}$$

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.

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 $\max_{J \in \mathcal{I}} |J \cap I| \le d$.

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) $\mathbb{P}[|I| \ge n] \;\ge\; 0.9$
(2) $\mathbb{P}[|I \cap J| \ge d] \;\le\; \frac{1}{2} \cdot 2^{-d/10}$ for any $J \in \mathcal{I}$

Since $|\mathcal{I}| \le 2^{d/10}$ and union bound on (2), $\mathbb{P}[\max_{J \in \mathcal{I}} |J \cap I| \ge d] \le \frac{1}{2}$.
Hence, with probability at least $0.9 \cdot 0.5 = 0.45$, our random set I has intersection $\le d$ with all old sets and $\ge n$ 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 $\ge 0.45^m > 0$, so some choice of sets $\mathcal{I}$ as claimed must exist. ■