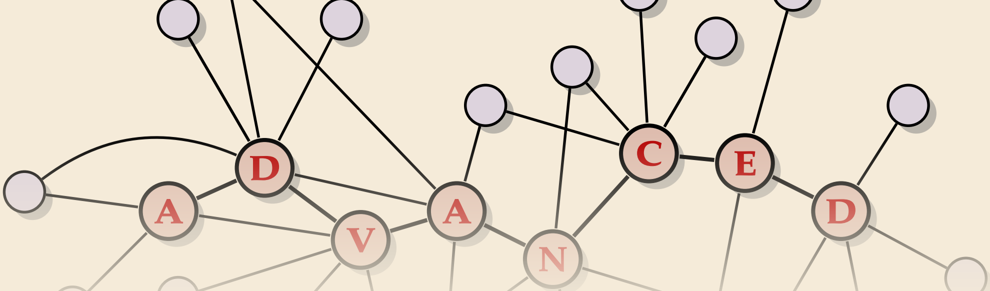


8 Randomized Complexity

18 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-17 00:25 H`

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?

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 Matter

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,

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

◀

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)

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

◀

⇝ 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) $\mathbb{P}[A(x) = 1] \ge \frac{1}{2}$ 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 $\mathsf{NP} \cap \mathsf{co\text{-}NP} \overset{?}{=} \mathsf{P}$.
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

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.

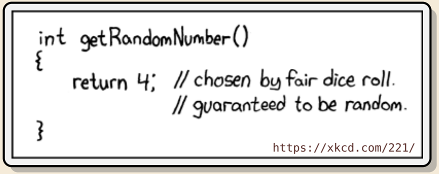
- **But:** If $Random_A(n) \le c \cdot \overset{= \log_2}{\lg}(n)$,
the derandomization of A runs in polytime: $n^c \cdot Time_A(n)$

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 (!)

Boolean Circuits Complexity

Formalization Pseudorandom Generator

8.3 Nisan-Wigderson Construction

Yao's Theorem

Nisan-Wigderson Construction

Combinatorial Designs

Probabilistic Method for Combinatorial Designs

8.4 Derandomization of BPP?

Pseudorandom Generator for BPP Derandomization