Randomized Algorithms

— P, NP, RP, PP, ZPP, BPP, ...

Joseph Chuang-Chieh Lin

Department of Computer Science & Engineering,
National Taiwan Ocean University

Spring 2026

Outline

1. RAMs & Turing Machines
2. Complexity Classes
   - Deterministic Classes
   - Space Complexity Classes
   - Reduction & Completeness
   - Randomized Complexity Classes
3. Transformation of Probability Distributions

Outline

RAM (Random Access Machine)

- RAM is a model of computation used when describing and analyzing algorithms.
- A machine can perform operations involving registers and main memory.
- The *unit-cost* RAM: each instruction can be performed in one time step.
    - Too powerful; no known polynomial time simulation of this type of model by Turing machines.
- The *log-cost* RAM: each instruction requires time proportional to the logarithm of the size of its operands.

Turing Machine

A physical Turing machine (with finite amount of tape).

Deterministic Turing Machine

A deterministic Turing machine is a quadruple $M = (S, \Sigma, \delta, s)$.

- S: a finite set of states (s: the initial state)
- Σ: A finite set of symbols (including special symbols BLANK and FIRST).
- δ: the transition function.
  - $S \times \Sigma \mapsto (S \cup \{\text{HALT, YES, NO}\}) \times \Sigma \times \{\leftarrow, \rightarrow, \text{STAY}\}$.
  - HALT, YES, NO: The three halting states not in S.

Turing Machine (Input & Tape)

- The input to the TM: written on a tape.
- The TM, as an algorithm, may read from and write on this tape.
- Assume that HALT, YES, NO as well as the symbols $\leftarrow, \rightarrow$, and STAY are not in $S \cup \Sigma$.
- The TM begins in the initial state s with its cursor at the first symbol FIRST of input x.
- The input is a string of $(\Sigma \setminus \{\text{BLANK}, \text{FIRST}\})^*$.
  - The left-most BLANK on the tape: the end of the input string.

Turing Machine (Transition)

- The transition function δ: can be thought as a *program*.
- In each step, the TM reads the symbol α pointed by the cursor;
- Based on α and the current state, choose:
    - a next state;
    - a symbol β to be overwritten on α;
    - a cursor motion direction from $\{\leftarrow, \rightarrow, \text{STAY}\}$.
- The cursor never falls off the left end of the input: FIRST.
- The BLANK symbol can be overwritten.

Turing Machine (Acceptance & Reject)

- The TM has accepted the input x: if the TM halts in the YES state.
- The TM has rejected the input x: if the TM halts in the NO state.
- State HALT: for the computation of functions whose range is not Boolean (output of the function is written on the tape).

Probabilistic Turing Machine

A probabilistic Turing machine is a Turing machine augmented with the ability to generate an unbiased coin flip in one step.

- This corresponds to a randomized algorithm.

Outline

SAT

An instance of satisfiability (SAT):

$$(x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_3 \wedge \neg x_4 \wedge x_5) \vee (\neg x_1 \wedge x_2 \wedge x_4 \wedge \neg x_5)$$

- $x_1, x_2, \ldots$: variables
- $\neg x_1, x_2$: literals
- $(\cdots)$: clauses

Language Recognition Problems

Language Recognition Problems

Any decision problem can be treated as a language recognition problem.

- Σ^*: the set of all possible strings over Σ.
- $|S|$: length of string s.

A language $L \subseteq \Sigma^*$ is any collection of strings over Σ.

A Language Recognition Problem

Decide whether a given string $x \in \Sigma^*$ belongs to L.

Complexity Class

A collection of languages all of whose recognition problems can be solved under prescribed bounds on the computational resources.

$\boldsymbol{P}$ & $\boldsymbol{NP}$

P

The class $\boldsymbol{P}$ consists of all languages L which has a polynomial time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow A(x)$ accepts;
- $x \notin L \Rightarrow A(x)$ rejects.

NP

The class $\boldsymbol{NP}$ consists of all languages L which has a polynomial time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \exists y \in \Sigma^*$, $A(x, y)$ accepts for $|y| \leq \text{poly}(|x|)$;
- $x \notin L \Rightarrow \forall y \in \Sigma^*$, $A(x, y)$ rejects.

For example, given an instance of satisfiability (SAT):

$$\mathbf{x} = (x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_3 \wedge \neg x_4 \wedge x_5) \vee (\neg x_1 \wedge x_2 \wedge x_4 \wedge \neg x_5)$$

For example, given an instance of satisfiability (SAT):

$$\mathbf{x} = (x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_3 \wedge \neg x_4 \wedge x_5) \vee (\neg x_1 \wedge x_2 \wedge x_4 \wedge \neg x_5)$$

Suppose we have the "proof" $\mathbf{y}$ as

$$x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{True}, x_4 = \text{True}, x_5 = \text{True}$$

For example, given an instance of satisfiability (SAT):

$$\mathbf{x} = (x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_3 \wedge \neg x_4 \wedge x_5) \vee (\neg x_1 \wedge x_2 \wedge x_4 \wedge \neg x_5)$$

Suppose we have the “proof” $\mathbf{y}$ as

$$x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{True}, x_4 = \text{True}, x_5 = \text{True}$$

We can check that $(\mathbf{x}, \mathbf{y}) \in L$ (encoded as string of $O(\log n)$ space in the tape) in polynomial time, where L denote the set of all satisfiable formula.

A Useful, Alternative Viewpoint

The class $\boldsymbol{P}$ consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be found and verified in polynomial time.

The class $\boldsymbol{NP}$ consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be verified in polynomial time.

Obviously,

$$\boldsymbol{P} \qquad \boldsymbol{NP}.$$

A Useful, Alternative Viewpoint

The class $\boldsymbol{P}$ consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be found and verified in polynomial time.

The class $\boldsymbol{NP}$ consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be verified in polynomial time.

Obviously,

$$\boldsymbol{P} \subseteq \boldsymbol{NP}.$$

Complementary Classes

For any complexity class $\mathcal{C}$, the complementary class co-$\mathcal{C}$ is the set of languages whose complement is in $\mathcal{C}$. That is,

$$\text{co-}\mathcal{C} = \{L \mid \bar{L} \in \mathcal{C}\}.$$

Examples: co-$\boldsymbol{P}$ & co-$\boldsymbol{NP}$

co-$\boldsymbol{P}$

The class co-$\boldsymbol{P}$ consists of all languages L which has a polynomial time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \notin L \Rightarrow A(x)$ accepts;
- $x \in L \Rightarrow A(x)$ rejects.

co-$\boldsymbol{NP}$

The class co-$\boldsymbol{NP}$ consists of all languages L which has a polynomial time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \notin L \Rightarrow \exists y \in \Sigma^*$, $A(x, y)$ accepts for $|y| \leq \text{poly}(|x|)$;
- $x \in L \Rightarrow \forall y \in \Sigma^*$, $A(x, y)$ rejects..

Open Questions: $\boldsymbol{P} = \boldsymbol{NP} \cap \text{co-}\boldsymbol{NP}$? $\quad \boldsymbol{NP} = \text{co-}\boldsymbol{NP}$?

Similarly, . . .

EXP & *NEXP*

EXP

The class ***EXP*** consists of all languages L which has an exponential time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow A(x)$ accepts;
- $x \notin L \Rightarrow A(x)$ rejects.

NEXP

The class ***NEXP*** consists of all languages L which has an exponential time algorithm A s.t. for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \exists y \in \Sigma^*$, $A(x, y)$ accepts for $|y| \leq \text{poly}(|x|)$;
- $x \notin L \Rightarrow \forall y \in \Sigma^*$, $A(x, y)$ rejects..

A Useful, Alternative Viewpoint

The class ***EXP*** consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be found and verified in exponential time.

The class ***NEXP*** consists of all language L such that for any $x \in L$, a proof of $x \in L$ (represented by the string y) can be verified in exponential time.

Obviously,

$$\boldsymbol{EXP} \subseteq \boldsymbol{NEXP}.$$

Outline

Space

- The space used by a TM: the number of distinct positions on the tape that are scanned during an execution.
    - For RAMs, its the number of words of memory required by an algorithm.

- ***PSPACE*** and ***NPSPACE***: resembles the settings of ***P*** and ***NP*** but requiring polynomial space.

- A ***PSPACE*** algorithm may run for super-polynomial time (e.g., $2^{\text{poly}(n)}$).
- Known results: ***PSPACE*** = ***NPSPACE***, ***PSPACE*** = co-***PSPACE***.
    - Savitch's theorem: a deterministic Turing machine can simulate a nondeterministic Turing machine without needing much more space.

Outline

1. RAMs & Turing Machines
2. Complexity Classes
   - Deterministic Classes
   - Space Complexity Classes
   - Reduction & Completeness
   - Randomized Complexity Classes
3. Transformation of Probability Distributions

Reduction

Polynomial Reduction

A polynomial reduction from a language $L_1 \subseteq \Sigma^*$ to a language $L_2 \subseteq \Sigma^*$ is a function $f : \Sigma^* \mapsto \Sigma^*$ such that

- $\exists$ a polynomial time algorithm that computes f
- $\forall x \in \Sigma^*$, $x_1 \in L_1$ if and only if $f(x) \in L_2$.

Completeness

$\boldsymbol{NP}$-hard

A language L is $\boldsymbol{NP}$-hard if, for all $L' \in \boldsymbol{NP}$, there is a polynomial reduction from L' to L.

$\boldsymbol{NP}$-complete

A language L is $\boldsymbol{NP}$-complete if it is in $\boldsymbol{NP}$ and is $\boldsymbol{NP}$-hard.

The first $\boldsymbol{NP}$-complete problem: SAT (Cook-Levin Theorem (1971)).

Outline

RP

RP

The class ***RP*** (i.e., Randomized Polynomial time) consists of all languages L that have a randomized algorithm A which runs in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

- Err only when $x \in L$. $\Rightarrow$ one-sided error.

co-*RP*

co-*RP*

The class co-***RP*** (i.e., complement Randomized Polynomial time) consists of all languages L that have a randomized algorithm A which runs in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$.
- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

- Err only when $x \notin L$. $\Rightarrow$ one-sided error.

Exercise (3%)

Assume that we have the following class:

$\boldsymbol{RP'}$

The class $\boldsymbol{RP'}$ consists of all languages L that have a randomized algorithm A which runs in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{1}{n^2}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

Prove that $\boldsymbol{RP'} = \boldsymbol{RP}$.

$\textbf{\textit{RP}} \cap \text{co-}\textbf{\textit{RP}}$

$\textbf{\textit{ZPP}}$ (Zero-error Probabilistic Polynomial time

The class $\textbf{\textit{ZPP}}$ is the class of languages that have Las Vegas algorithms running in expected polynomial time.

Why *ZPP*?

- Suppose we have a language $L \in \textbf{\textit{RP}} \cap \text{co-}\textbf{\textit{RP}}$.
- L can be recognized by an ***RP*** algorithm A and a co-***RP*** algorithm B.

A Las Vegas algorithm

Given the input x, perform the following procedure in iterations.

1. If $A(x)$ accepts, then x must be a YES-instance;
2. Otherwise, if $B(x)$ rejects, then x must be a NO-instance.
3. If neither of above occurs, continue to next iteration.

- The expected number of iterations is bounded!

PP

PP

The class ***PP*** (i.e., Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] > \frac{1}{2}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] < \frac{1}{2}$.

PP

PP

The class ***PP*** (i.e., Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] > \frac{1}{2}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] < \frac{1}{2}$.

- To reduce the error probability of a two-sided error algorithm, we can perform several independent iterations on the same input.
- Output the majority answer of these iterations.

BPP

BPP

The class ***BPP*** (i.e., Bounded-error Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \leq \frac{1}{4}$.

BPP

BPP

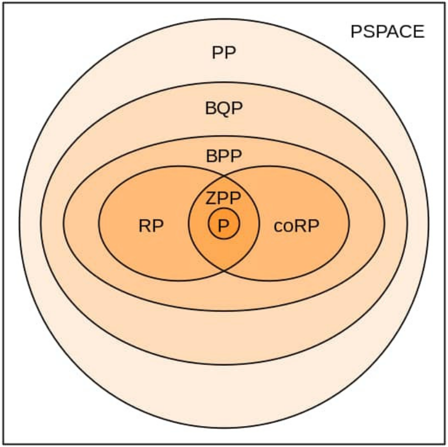
The class ***BPP*** (i.e., Bounded-error Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \leq \frac{1}{4}$.

- To reduce the error probability of a two-sided error algorithm, we can perform several independent iterations on the same input.
- Output the majority answer of these iterations.

Randomized Complexity Classes

Source: Wikipedia



Outline

p-coin & Transforming a Probability Distribution

p-coin

A coin is called a p-coin if it shows HEAD after one coin-flipping with probability p.

Probability Distribution Transformations

A function that transforms a p-coin to get a q-coin, for $0 < p, q < 1$, is called a p-to-q transformation.

p-coin & Transforming a Probability Distribution

p-coin

A coin is called a p-coin if it shows HEAD after one coin-flipping with probability p.

Probability Distribution Transformations

A function that transforms a p-coin to get a q-coin, for $0 < p, q < 1$, is called a p-to-q transformation.

Easy cases:

- p-to-0 and p-to-1..

p-coin & Transforming a Probability Distribution

p-coin

A coin is called a p-coin if it shows HEAD after one coin-flipping with probability p.

Probability Distribution Transformations

A function that transforms a p-coin to get a q-coin, for $0 < p, q < 1$, is called a p-to-q transformation.

Easy cases:

- p-to-0 and p-to-1..
- p-to-$(1 - p)$

p-coin & Transforming a Probability Distribution

p-coin

A coin is called a p-coin if it shows HEAD after one coin-flipping with probability p.

Probability Distribution Transformations

A function that transforms a p-coin to get a q-coin, for $0 < p, q < 1$, is called a p-to-q transformation.

Easy cases:

- p-to-0 and p-to-1..
- p-to-$(1-p)$
- p-to-p^2.

p-coin & Transforming a Probability Distribution

p-coin

A coin is called a p-coin if it shows HEAD after one coin-flipping with probability p.

Probability Distribution Transformations

A function that transforms a p-coin to get a q-coin, for $0 < p, q < 1$, is called a p-to-q transformation.

Easy cases:

- p-to-0 and p-to-1..
- p-to-$(1-p)$
- p-to-p^2.
- p-to-$(\binom{n}{k}p^k(1-p)^{n-k})$, for $n \in \mathbb{N}$ and $k \in \{0, 1, \ldots, n\}$.

Exercise (2%)

Given a p-coin, where $0 < p < 1$.

- Repeat the following steps until it returns YES or NO.
  1. flip the p-coin twice.
  2. if the results are HEAD-TAIL, return YES;
  3. else if the results are TAIL-HEAD, return NO;
  4. otherwise, continue to next iteration

- Please prove that the above procedure is a p-to-$\frac{1}{2}$ transformation (i.e., deriving a fair coin).
- Please compute the expected number of coin-flips of the above procedure.

Concatenation Rule

Concatenation Rule

Given a p-coin and a q-coin, we can derive a pq-coin as follows.

1. First, simulate the p-coin. If it is TAIL, output TAIL
2. Otherwise, simulate the q-coin and output the outcome.

Concatenation Rule

Concatenation Rule

Given a p-coin and a q-coin, we can derive a pq-coin as follows.

1. First, simulate the p-coin. If it is TAIL, output TAIL
2. Otherwise, simulate the q-coin and output the outcome.

- By the Concatenation Rule & the exercise, we can derive a $p/2$-coin when we are given a p-coin.

Discussions