

# Design and Analysis of Algorithm

## Basics of Complexity Theory

- 1 Decision Problem
- 2 Deterministic Computation
- 3 Several Important Complexity Classes
  - $\mathcal{P}$  vs.  $\mathcal{NP}$
  - $\mathcal{NP}$ -complete
  - co- $\mathcal{NP}$
- 4 Randomized Computation
  - $\mathcal{BPP}$
  - PSPACE
- 5 Decision vs. Search
- 6 Impact on Cryptography

# Outline

## 1 Decision Problem

## 2 Deterministic Computation

## 3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

## 4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

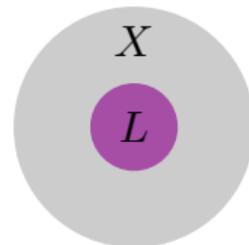
## 5 Decision vs. Search

## 6 Impact on Cryptography

## Decision Problem

Decision Problem: recognition of a set of strings  $L \subseteq X$

- $X$ : a set of strings
- $x$ : a string in  $X$  (each string corresponds to an instance)
- $L$ : language (a subset of  $X$  satisfying some property)

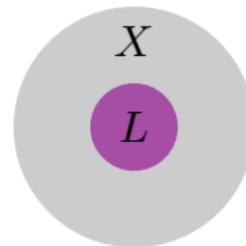


Task: Decide membership — if  $x \in L$

## Decision Problem

Decision Problem: recognition of a set of strings  $L \subseteq X$

- $X$ : a set of strings
- $x$ : a string in  $X$  (each string corresponds to an instance)
- $L$ : language (a subset of  $X$  satisfying some property)



Task: Decide membership — if  $x \in L$

### Example

- $X = \mathbb{N}$
- $L$  are Primes =  $\{2, 3, 5, 7, 11, 13, \dots\}$
- decide if  $x$  is a prime.

## Motivation for Complexity Theory

We always want to know if a given problem can be *efficiently* solved by an algorithm.

## Motivation for Complexity Theory

We always want to know if a given problem can be *efficiently* solved by an algorithm.

- ① Precisely model algorithms
  - What is computation?
  - What is computable?
- ② Precisely define what does it means for efficient.

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Turing Machine

1936, London Mathematical Society: On computable numbers, with an application to the Entscheidungs Problem.

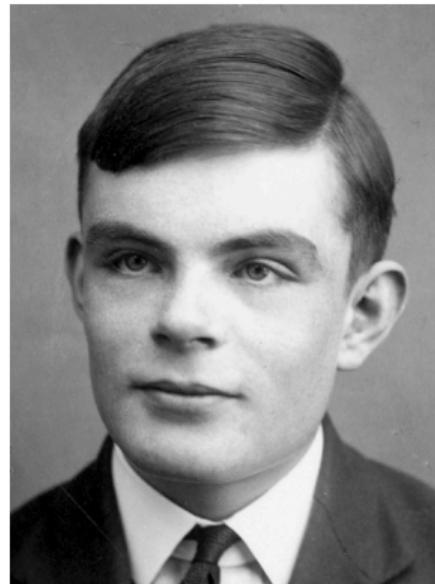
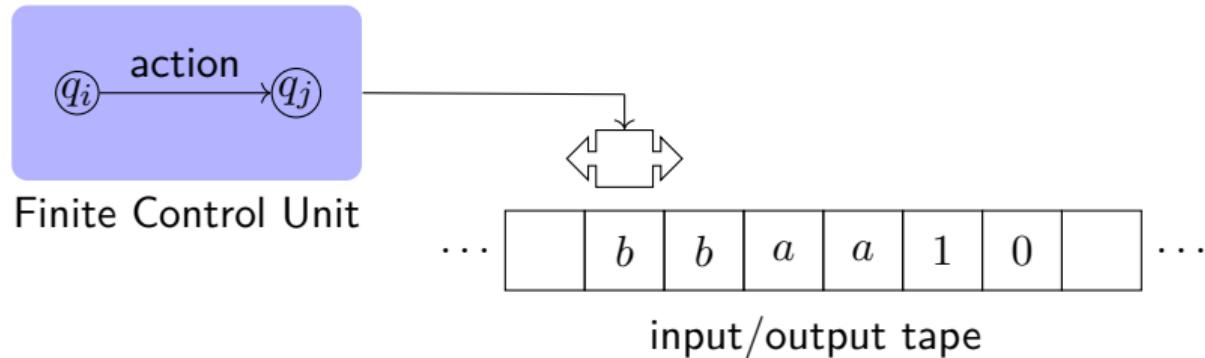


Figure: Alan Turing

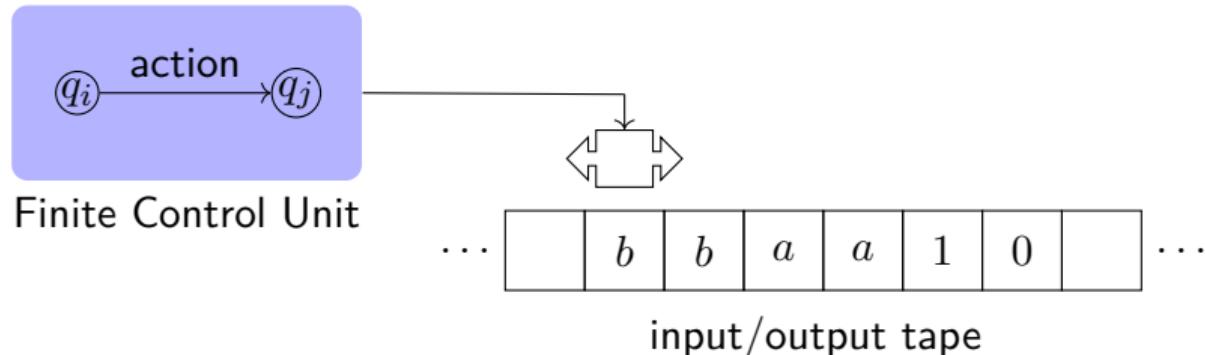
## Turing Machine

**Turing machine:** automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.



## Turing Machine

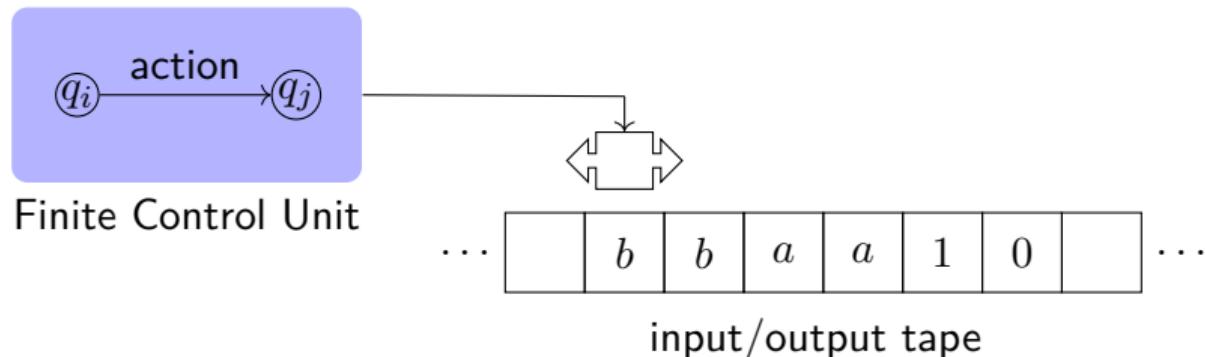
**Turing machine:** automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.



- At the beginning, the tape contains the input in several cells. Other places are empty.

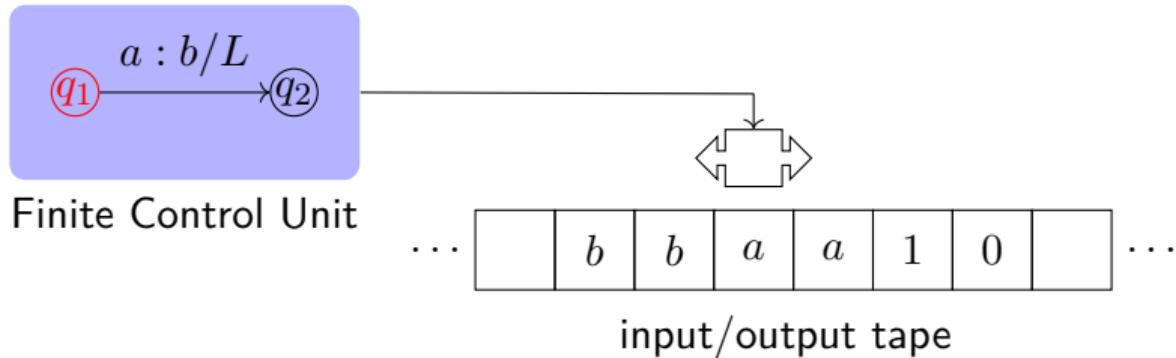
## Turing Machine

**Turing machine:** automatic machine that has a tape (divided into infinite cells), a control unit and a read/write head.

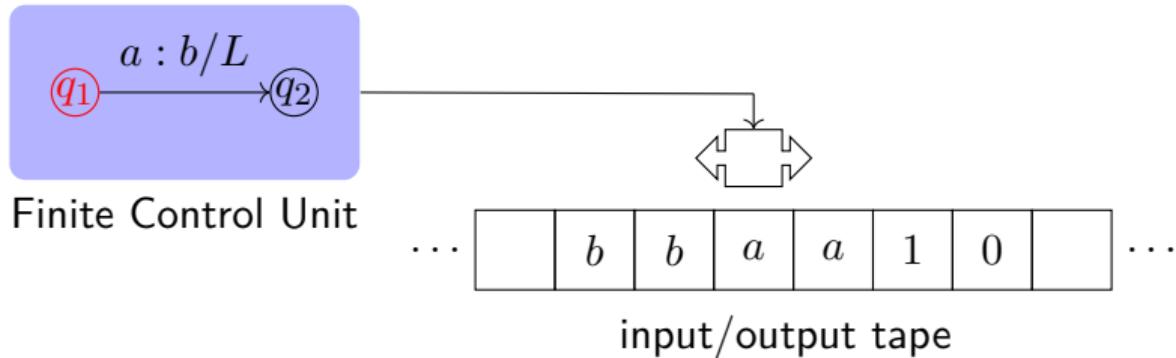


- At the beginning, the tape contains the input in several cells. Other places are empty.
- During computation, the control unit monitor current state and the head value, can do the following operations:
  - ① wipe off old value and write new values
  - ② change the current state
  - ③ move head left or right

## An Example

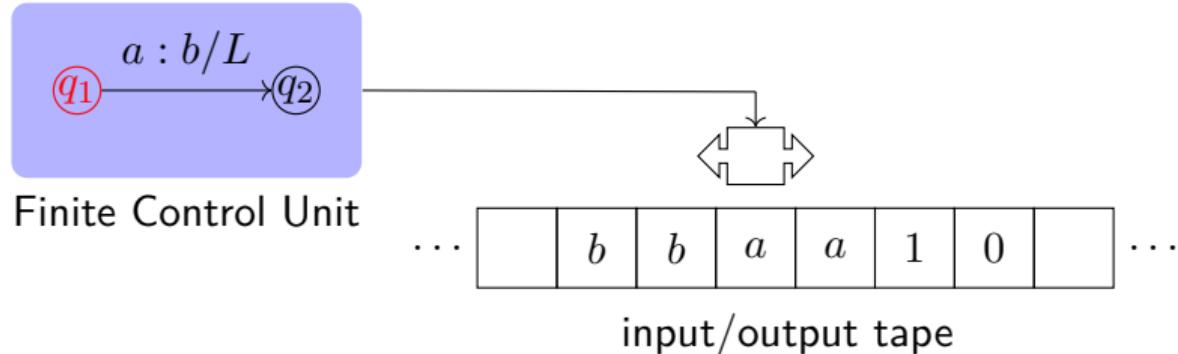


## An Example

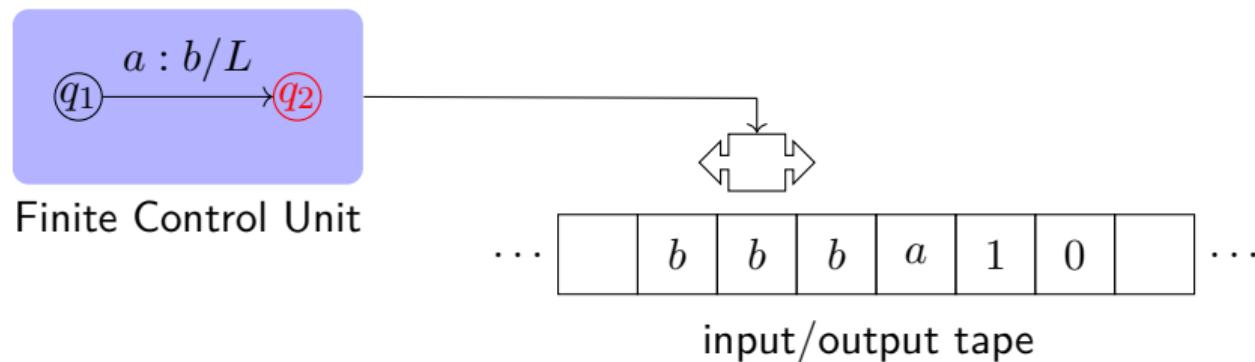


- $a \rightarrow b$ ; move left; current state  $q_1 \rightarrow q_2$

## An Example



- $a \rightarrow b$ ; move left; current state  $q_1 \rightarrow q_2$



## Intuition of Turing Machine

Mimic how human being solve a problem

## Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)

## Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)
- TM is provided a tape, which contains infinite cells (paper)

## Intuition of Turing Machine

Mimic how human being solve a problem

- TM has a finite number of states (memory)
- TM is provided a tape, which contains infinite cells (paper)
- a symbol can be scanned from a cell or printed to a cell (reading and writing)

## Formal Definition

### Definition 1 (Turing Machine)

TM consists  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$

- $Q$ : a finite set of states
- $\Sigma$ : input alphabets
- $\Gamma$ : working alphabets (including  $\perp$ ,  $\Sigma \subseteq \Gamma$ )
- $q_0$ : the initial state of  $Q$ ;
- $q_{\text{acc}}, q_{\text{rej}}$ : accept and reject state of  $Q$
- $\delta$ : transition function

$$\delta : (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

## Running Time of TM

### Definition 2

We denote the running time of TM by  $t_M(n)$ , which is the maximum steps that TM runs on all inputs of length  $n$

### Polynomial Time

$$\bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

## The Extended Church-Turing Thesis

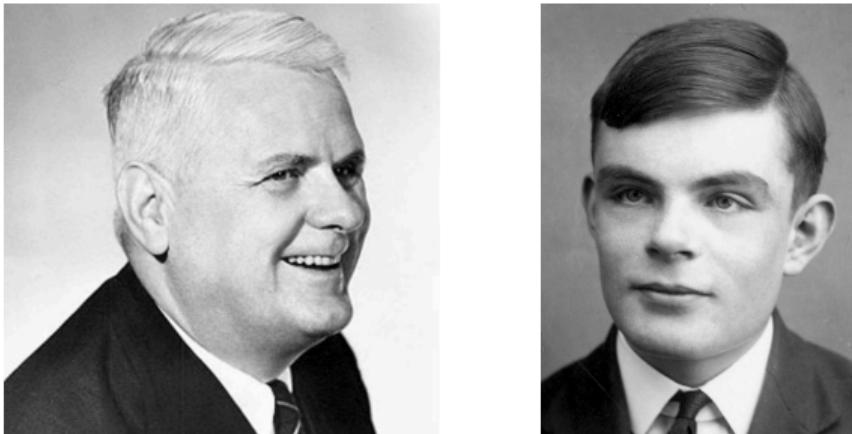


Figure: Alonzo Church & Alan Turing

Everyone's intuition of **Efficient** Algorithms = **Polynomial-Time** deterministic TMs

## Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

## Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

- ① NDTM may proceed according to [several possible transitions](#)
  - We can assume that every configuration leads to two possible configurations.

## Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

- ① NDTM may proceed according to **several possible transitions**
  - We can assume that every configuration leads to two possible configurations.
- ② NDTM *accepts* iff there exists **at least one of its branches accepts**.

## Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

- ① NDTM may proceed according to **several possible transitions**
  - We can assume that every configuration leads to two possible configurations.
- ② NDTM *accepts* iff there exists **at least one of its branches accepts**.

NDTM doesn't really correspond to any real-world physical model, it's just a theoretical construction.

## Non-deterministic Turing Machine

Non-deterministic TMs are just like standard TMs, except:

- ① NDTM may proceed according to **several possible transitions**
  - We can assume that every configuration leads to two possible configurations.
- ② NDTM *accepts* iff there exists **at least one of its branches accepts**.

NDTM doesn't really correspond to any real-world physical model, it's just a theoretical construction.

Non-determinism doesn't give TM any power to recognize more languages.

- Any NDTM can be simulated by a TM (with potentially exponential time overhead) by trying all branches of the NDTM machine “in parallel” by using BFS.

## Notes

Some important facts about TM

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .
- ② Any  $\{0, 1\}^*$  represents some TM.

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .
- ② Any  $\{0, 1\}^*$  represents some TM.
- ③ Any TM has many representations in the form of  $\{0, 1\}^*$ .

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .
- ② Any  $\{0, 1\}^*$  represents some TM.
- ③ Any TM has many representations in the form of  $\{0, 1\}^*$ .

*Why TMs are so powerful?*

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .
- ② Any  $\{0, 1\}^*$  represents some TM.
- ③ Any TM has many representations in the form of  $\{0, 1\}^*$ .

*Why TMs are so powerful?*

- TM has a working tape (好记性不如烂笔头)

Some important facts about TM

- ① Any TM can be encoded as  $\{0, 1\}^*$ .
- ② Any  $\{0, 1\}^*$  represents some TM.
- ③ Any TM has many representations in the form of  $\{0, 1\}^*$ .

*Why TMs are so powerful?*

- TM has a working tape (好记性不如烂笔头)
- TM itself can be treated as data! TM can take another TM as its input.

## Universal TM



## Universal TM



## Universal TM



||



# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Time Complexity Hierarchy: $\mathcal{P}$ and $\mathcal{NP}$

We have introduced

- the notion of decision problems  $L \subseteq X$
- the concept of (non-deterministic) TM

## Time Complexity Hierarchy: $\mathcal{P}$ and $\mathcal{NP}$

We have introduced

- the notion of decision problems  $L \subseteq X$
- the concept of (non-deterministic) TM

We say a TM  $M$  accepts (“1” signify acceptance) a language  $L$  if:

$$x \in L \iff M(x) = 1$$

- $L$  is decidable by  $M$  ( $M$  solves  $L$ )

## Time Complexity Hierarchy: $\mathcal{P}$ and $\mathcal{NP}$

We have introduced

- the notion of decision problems  $L \subseteq X$
- the concept of (non-deterministic) TM

We say a TM  $M$  accepts ("1" signify acceptance) a language  $L$  if:

$$x \in L \iff M(x) = 1$$

- $L$  is decidable by  $M$  ( $M$  solves  $L$ )
- 

Next, we introduce two important sets of problems, characterized by time complexity by DTM and NDTM:

$\mathcal{P}$  and  $\mathcal{NP}$

### Definition 3 ( $\mathcal{P}$ Language)

$L \in \mathcal{P}$  if there exists a deterministic poly-time TM  $M$  such that

$$M(x) = 1 \iff x \in L$$

### Definition 3 ( $\mathcal{P}$ Language)

$L \in \mathcal{P}$  if there exists a deterministic poly-time TM  $M$  such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most  $p(n)$  steps, where  $p(\cdot)$  is some polynomial and  $n$  is the length of input

### Definition 3 ( $\mathcal{P}$ Language)

$L \in \mathcal{P}$  if there exists a deterministic poly-time TM  $M$  such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most  $p(n)$  steps, where  $p(\cdot)$  is some polynomial and  $n$  is the length of input

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

## $\mathcal{P}$ Complexity

### Definition 3 ( $\mathcal{P}$ Language)

$L \in \mathcal{P}$  if there exists a deterministic poly-time TM  $M$  such that

$$M(x) = 1 \iff x \in L$$

- poly-time: take at most  $p(n)$  steps, where  $p(\cdot)$  is some polynomial and  $n$  is the length of input

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

### Example of $\mathcal{P}$ Languages

- $L = \{\text{even integers}\}$ ,  $M$  just need to check if the last bit is 0.
- $L = \text{PRIME}$ ,  $M$  is the AKS primality test algorithm.

## $\mathcal{NP}$ Complexity

Definition 4 ( $\mathcal{NP}$  Languages - Conventional)

$L \in \mathcal{NP}$  if there exists a non-deterministic poly-time TM  $M$ :

$$x \in L \iff M(x) = 1$$

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

### Definition 4 ( $\mathcal{NP}$ Languages - Conventional)

$L \in \mathcal{NP}$  if there exists a non-deterministic poly-time TM  $M$ :

$$x \in L \iff M(x) = 1$$

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

### Warning

$\mathcal{NP}$  means non-deterministic poly-time, not **non-poly-time!**

## Modern Definition

Definition 5 ( $\mathcal{NP}$  Complexity - Modern)

$L \in \mathcal{NP}$  if there exists a deterministic poly-time TM  $M$ :

$$x \in L \iff \exists w \in \{0, 1\}^{\text{poly}(n)} \text{ s.t. } M(x, w) = 1$$

## Modern Definition

Definition 5 ( $\mathcal{NP}$  Complexity - Modern)

$L \in \mathcal{NP}$  if there exists a deterministic poly-time TM  $M$ :

$$x \in L \iff \exists w \in \{0, 1\}^{\text{poly}(n)} \text{ s.t. } M(x, w) = 1$$

- If  $M(x, w) = 1$ ,  $w$  is called as a “witness” for  $x \in L$ . One can think of  $w$  as an efficiently-verifiable “certificate” or “proof” for  $x \in L$ .

## Modern Definition

### Definition 5 ( $\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$  if there exists a deterministic poly-time TM  $M$ :

$$x \in L \iff \exists w \in \{0, 1\}^{\text{poly}(n)} \text{ s.t. } M(x, w) = 1$$

- If  $M(x, w) = 1$ ,  $w$  is called as a “witness” for  $x \in L$ . One can think of  $w$  as an efficiently-verifiable “certificate” or “proof” for  $x \in L$ .
- We require  $w$  is *short*, i.e.,  $|w| = \text{poly}(n)$ . This is natural since  $M$  runs in poly-time in  $n$ .

## Modern Definition

### Definition 5 ( $\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$  if there exists a deterministic poly-time TM  $M$ :

$$x \in L \iff \exists w \in \{0, 1\}^{\text{poly}(n)} \text{ s.t. } M(x, w) = 1$$

- If  $M(x, w) = 1$ ,  $w$  is called as a “witness” for  $x \in L$ . One can think of  $w$  as an efficiently-verifiable “certificate” or “proof” for  $x \in L$ .
- We require  $w$  is *short*, i.e.,  $|w| = \text{poly}(n)$ . This is natural since  $M$  runs in poly-time in  $n$ .

$L \in \mathcal{NP}$  iff there are *short proofs* for membership of every elements in  $L$ .

## Modern Definition

### Definition 5 ( $\mathcal{NP}$ Complexity - Modern)

$L \in \mathcal{NP}$  if there exists a deterministic poly-time TM  $M$ :

$$x \in L \iff \exists w \in \{0, 1\}^{\text{poly}(n)} \text{ s.t. } M(x, w) = 1$$

- If  $M(x, w) = 1$ ,  $w$  is called as a “witness” for  $x \in L$ . One can think of  $w$  as an efficiently-verifiable “certificate” or “proof” for  $x \in L$ .
- We require  $w$  is *short*, i.e.,  $|w| = \text{poly}(n)$ . This is natural since  $M$  runs in poly-time in  $n$ .

$L \in \mathcal{NP}$  iff there are *short proofs* for membership of every elements in  $L$ .

### Equivalence between traditional and modern definitions

- Even though  $M$  is a deterministic machine, its second argument  $w$  captures the nondeterminism in the definition.

## Examples of $\mathcal{NP}$ Language - Composites

$L = \text{COMPOSITE}$

- instance  $x$  is an integer
- a witness  $w$  for  $x \in L$  is a non-trivial factor of  $x$
- $M$  just need to check if  $w$  divides  $x$ , which could be done in polynomial time.

## Examples of $\mathcal{NP}$ Language - Composites

$L = \text{COMPOSITE}$

- instance  $x$  is an integer
- a witness  $w$  for  $x \in L$  is a non-trivial factor of  $x$
- $M$  just need to check if  $w$  divides  $x$ , which could be done in polynomial time.

### Example of COMPOSITE

- instance: 15
- witness: 3, 5

## Examples of $\mathcal{NP}$ Language - Composites

$L = \text{COMPOSITE}$

- instance  $x$  is an integer
- a witness  $w$  for  $x \in L$  is a non-trivial factor of  $x$
- $M$  just need to check if  $w$  divides  $x$ , which could be done in polynomial time.

Example of COMPOSITE

- instance: 15
- witness: 3, 5

In fact, COMPOSITE also belong to  $\mathcal{P}$  (think why?)

## Examples of $\mathcal{NP}$ Language - Composites

$L = \text{COMPOSITE}$

- instance  $x$  is an integer
- a witness  $w$  for  $x \in L$  is a non-trivial factor of  $x$
- $M$  just need to check if  $w$  divides  $x$ , which could be done in polynomial time.

### Example of COMPOSITE

- instance: 15
- witness: 3, 5

In fact, COMPOSITE also belong to  $\mathcal{P}$  (think why?)

$\mathcal{P}$  is closed under complement, i.e.,  $L \in \mathcal{P} \Rightarrow \overline{L} \in \mathcal{P}$ .

- A deterministic TM can simply flip the accept/reject output at the end, and this does not change the running time.

## Examples of $\mathcal{NP}$ Language - SAT and 3-SAT

SAT: Given a CNF formula  $\Phi$ , check if it has a satisfying truth assignment.

3-SAT: SAT where each clause contains exactly 3 literals

witness: an assignment of truth values to the Boolean variables

## Examples of $\mathcal{NP}$ Language - SAT and 3-SAT

SAT: Given a CNF formula  $\Phi$ , check if it has a satisfying truth assignment.

3-SAT: SAT where each clause contains exactly 3 literals

witness: an assignment of truth values to the Boolean variables

### Example of 3-SAT

- instance  $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
- witness:  $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$

## Examples of $\mathcal{NP}$ Language - Hamilton Path

**Hamilton Graph:** Given an undirected graph  $G = (V, E)$ , does there exist a simple path that visits every node?

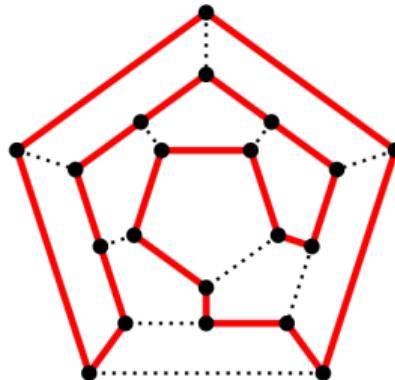


Figure: Hamiltonian Graph (a path traverses through each vertex exactly once)

witness: a path

$M$  check if the path contains each node in  $V$  exactly once

## $\mathcal{P}$ vs. $\mathcal{NP}$

As per definition,  $\mathcal{P} \subseteq \mathcal{NP}$ . Because  $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$ :

- $M'(x, w)$  can always set  $w = \perp$  and decide whether  $x \in L$  using  $M$ .
- Alternatively, “short”  $M$  can be viewed as a witness for  $x \in L$ . Think about why the description of  $M$  is short?

## $\mathcal{P}$ vs. $\mathcal{NP}$

As per definition,  $\mathcal{P} \subseteq \mathcal{NP}$ . Because  $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$ :

- $M'(x, w)$  can always set  $w = \perp$  and decide whether  $x \in L$  using  $M$ .
- Alternatively, “short”  $M$  can be viewed as a witness for  $x \in L$ . Think about why the description of  $M$  is short?

$\mathcal{P}$  = the set of decision problems whose all instances can be efficiently decided

$\mathcal{NP}$  = the set of decision problems whose yes instances can be efficiently decided with short proofs

## $\mathcal{P}$ vs. $\mathcal{NP}$

As per definition,  $\mathcal{P} \subseteq \mathcal{NP}$ . Because  $L \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$ :

- $M'(x, w)$  can always set  $w = \perp$  and decide whether  $x \in L$  using  $M$ .
- Alternatively, “short”  $M$  can be viewed as a witness for  $x \in L$ . Think about why the description of  $M$  is short?

$\mathcal{P}$  = the set of decision problems whose all instances can be efficiently decided

$\mathcal{NP}$  = the set of decision problems whose yes instances can be efficiently decided with short proofs

1971: Cook, Edmonds, Levin, Yablonski, Gödel

Perhaps the most prominent question in TCS:

$$\mathcal{P} = ?\mathcal{NP}$$

$$\mathcal{P} = \mathcal{N}\mathcal{P}$$

$$\mathcal{P} = \mathcal{N}\mathcal{P}$$

If  $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!



生活在经验里  
直到大厦崩塌

If  $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!



- Cryptography as we know it may be impossible.
- Cryptographic researchers are out of job.

If  $\mathcal{P} = \mathcal{NP}$

The foundation of modern cryptography collapse!



- Cryptography as we know it may be impossible.
- Cryptographic researchers are out of job.

In principle, every aspect of life could be efficiently and globally optimized . . .  
    . . . life as we know it would be different!

## The Consequence of $\mathcal{P} = \mathcal{NP}$

$\mathcal{P} = \mathcal{NP} \Rightarrow$  OWF does not exist

## The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . To efficiently find a pre-image  $x$  of  $y$ , the idea is to determine  $x$  bit-by-bit.  $f(x_1 || \dots || x_n) = y$ .

## The Consequence of $\mathcal{P} = \mathcal{NP}$

$\mathcal{P} = \mathcal{NP} \Rightarrow$  OWF does not exist

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . To efficiently find a pre-image  $x$  of  $y$ , the idea is to determine  $x$  bit-by-bit.  $f(x_1 || \dots || x_n) = y$ .

Define a collection of languages  $L_i = \{(y, z) | \exists w \text{ s.t. } y = f(z||w)\}$ , where  $z \in \{0, 1\}^i$ ,  $w \in \{0, 1\}^{n-i}$

- $L_i \in \mathcal{NP}$  and thus also belong to  $\mathcal{P}$  by assumption, we define **Invert** as:

## The Consequence of $\mathcal{P} = \mathcal{NP}$

$$\mathcal{P} = \mathcal{NP} \Rightarrow \text{OWF does not exist}$$

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . To efficiently find a pre-image  $x$  of  $y$ , the idea is to determine  $x$  bit-by-bit.  $f(x_1 || \dots || x_n) = y$ .

Define a collection of languages  $L_i = \{(y, z) | \exists w \text{ s.t. } y = f(z||w)\}$ , where  $z \in \{0, 1\}^i$ ,  $w \in \{0, 1\}^{n-i}$

- $L_i \in \mathcal{NP}$  and thus also belong to  $\mathcal{P}$  by assumption, we define **Invert** as:

---

### Algorithm 4: Invert( $y$ )

---

```
1:  $z = \epsilon$ ;
2: for  $i \leftarrow 1$  to  $n$  do
3:   if  $(y, z||0) \in L_i$  then  $z = z||0$ ;
4:   else  $z = z||1$ ;
5: end
6: return  $z$ 
```

---

## The Reverse Direction

OWF exists  $\Rightarrow \mathcal{P} \neq \mathcal{NP}$

- We have many candidates of OWFs, but they require assumptions.

## The Reverse Direction

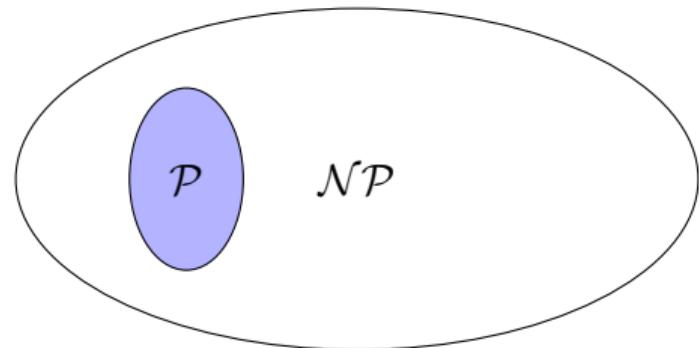
OWF exists  $\Rightarrow \mathcal{P} \neq \mathcal{NP}$

- We have many candidates of OWFs, but they require assumptions.

### Warning

OWFs do not exist *does not imply*  $\mathcal{P} = \mathcal{NP}$

## Consensus



## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

- Because  $\mathcal{NP}$  contains many languages that are not believed to be in  $\mathcal{P}$ , such as: 3-SAT and Hamiltonian graph.

## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

- Because  $\mathcal{NP}$  contains many languages that are not believed to be in  $\mathcal{P}$ , such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with  $n$  variables?

## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

- Because  $\mathcal{NP}$  contains many languages that are not believed to be in  $\mathcal{P}$ , such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with  $n$  variables?

Exhaustive search: try all  $2^n$  truth assignments.

## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

- Because  $\mathcal{NP}$  contains many languages that are not believed to be in  $\mathcal{P}$ , such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with  $n$  variables?

Exhaustive search: try all  $2^n$  truth assignments.

Q: Can we do anything substantially more clever?

## Evidence for $\mathcal{P} \subset \mathcal{NP}$

Consensus opinion:  $\mathcal{P} \neq \mathcal{NP}$

- Because  $\mathcal{NP}$  contains many languages that are not believed to be in  $\mathcal{P}$ , such as: 3-SAT and Hamiltonian graph.

Q: How to solve an instance of 3-SAT with  $n$  variables?

Exhaustive search: try all  $2^n$  truth assignments.

Q: Can we do anything substantially more clever?

Conjecture: No poly-time algorithm for 3-SAT  
intractable

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Motivation of Reduction

$\mathcal{NP}$  is the set of many problems.

*How to figure out the relations among them?*

A central approach is finding reductions

## Polynomial Time Reducibility

Language  $L'$  is *poly-time reducible* or *reduces* to language  $L$ , written as  $L' \leq_p L$ , if there is a deterministic poly-time function  $\mathcal{R} : L' \rightarrow L$  so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

## Polynomial Time Reducibility

Language  $L'$  is *poly-time reducible* or *reduces* to language  $L$ , written as  $L' \leq_p L$ , if there is a deterministic poly-time function  $\mathcal{R} : L' \rightarrow L$  so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$  is called a poly-time reduction from  $L$  to  $L'$ .

## Polynomial Time Reducibility

Language  $L'$  is *poly-time reducible* or *reduces* to language  $L$ , written as  $L' \leq_p L$ , if there is a deterministic poly-time function  $\mathcal{R} : L' \rightarrow L$  so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$  is called a poly-time reduction from  $L$  to  $L'$ .

$L' \leq_p L$  implies  $L'$  is not harder than  $L$

## Polynomial Time Reducibility

Language  $L'$  is *poly-time reducible* or *reduces* to language  $L$ , written as  $L' \leq_p L$ , if there is a deterministic poly-time function  $\mathcal{R} : L' \rightarrow L$  so that:

$$x \in L' \iff \mathcal{R}(x) \in L$$

$\mathcal{R}$  is called a poly-time reduction from  $L$  to  $L'$ .

$L' \leq_p L$  implies  $L'$  is not harder than  $L$

We should pay attention to:

- the direction of  $\mathcal{R}$
- the time complexity of  $\mathcal{R}$

### Definition 6 ( $\mathcal{NP}$ -Hard)

$L$  is said to be  $\mathcal{NP}$ -hard if for every  $\mathcal{NP}$ -language  $L'$ , there is a deterministic poly-time algorithm (a reduction)  $\mathcal{R}$ :

$$x \in L' \iff \mathcal{R}(x) \in L.$$

### Definition 6 ( $\mathcal{NP}$ -Hard)

$L$  is said to be  $\mathcal{NP}$ -hard if for every  $\mathcal{NP}$ -language  $L'$ , there is a deterministic poly-time algorithm (a reduction)  $\mathcal{R}$ :

$$x \in L' \iff \mathcal{R}(x) \in L.$$

- We can interpret that the languages in  $\mathcal{NP}$  is not harder than that in  $\mathcal{NP}$ -hard.

### Definition 6 ( $\mathcal{NP}$ -Hard)

$L$  is said to be  $\mathcal{NP}$ -hard if for every  $\mathcal{NP}$ -language  $L'$ , there is a deterministic poly-time algorithm (a reduction)  $\mathcal{R}$ :

$$x \in L' \iff \mathcal{R}(x) \in L.$$

- We can interpret that the languages in  $\mathcal{NP}$  is not harder than that in  $\mathcal{NP}$ -hard.

Fact: languages in  $\mathcal{NP}$ -hard may **not** fall in  $\mathcal{NP}$ .

### Definition 7 ( $\mathcal{NP}$ -Complete)

$L$  is  $\mathcal{NP}$ -complete if it is  $\mathcal{NP}$ -hard, and is itself in  $\mathcal{NP}$ .

### Definition 7 ( $\mathcal{NP}$ -Complete)

$L$  is  $\mathcal{NP}$ -complete if it is  $\mathcal{NP}$ -hard, and is itself in  $\mathcal{NP}$ .

**Definition Intuition:**  $\mathcal{NP}$ -complete represents the set of hardest problems in  $\mathcal{NP}$ .

### Definition 7 ( $\mathcal{NP}$ -Complete)

$L$  is  $\mathcal{NP}$ -complete if it is  $\mathcal{NP}$ -hard, and is itself in  $\mathcal{NP}$ .

**Definition Intuition:**  $\mathcal{NP}$ -complete represents the set of hardest problems in  $\mathcal{NP}$ .

- We can solve all problems in  $\mathcal{NP}$  if we find an efficient algorithm for any problems in  $\mathcal{NP}$ -complete.

## The Meaning of $\mathcal{NP}$ -complete

### Theorem 8

Suppose  $Y \in \mathcal{NP}$ -complete, then  $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$ .

## The Meaning of $\mathcal{NP}$ -complete

### Theorem 8

Suppose  $Y \in \mathcal{NP}$ -complete, then  $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$ .

$\Leftarrow$ :  $Y \in \mathcal{NP}$ -complete and thus of course  $Y \in \mathcal{NP}$ . Now suppose  $\mathcal{P} = \mathcal{NP}$ , we have  $Y \in \mathcal{P}$ .

## The Meaning of $\mathcal{NP}$ -complete

### Theorem 8

Suppose  $Y \in \mathcal{NP}$ -complete, then  $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$ .

$\Leftarrow$ :  $Y \in \mathcal{NP}$ -complete and thus of course  $Y \in \mathcal{NP}$ . Now suppose  $\mathcal{P} = \mathcal{NP}$ , we have  $Y \in \mathcal{P}$ .

$\Rightarrow$ :  $\forall X \in \mathcal{NP}$ ,  $X \leq_p Y$  because  $Y \in \mathcal{NP}$ -complete. Now suppose  $Y \in \mathcal{P}$ , we further have  $\mathcal{NP} \subseteq \mathcal{P}$ . We already know  $\mathcal{P} \subseteq \mathcal{NP}$ , thus  $\mathcal{P} = \mathcal{NP}$ .

## The Meaning of $\mathcal{NP}$ -complete

### Theorem 8

Suppose  $Y \in \mathcal{NP}$ -complete, then  $Y \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$ .

$\Leftarrow$ :  $Y \in \mathcal{NP}$ -complete and thus of course  $Y \in \mathcal{NP}$ . Now suppose  $\mathcal{P} = \mathcal{NP}$ , we have  $Y \in \mathcal{P}$ .

$\Rightarrow$ :  $\forall X \in \mathcal{NP}, X \leq_p Y$  because  $Y \in \mathcal{NP}$ -complete. Now suppose  $Y \in \mathcal{P}$ , we further have  $\mathcal{NP} \subseteq \mathcal{P}$ . We already know  $\mathcal{P} \subseteq \mathcal{NP}$ , thus  $\mathcal{P} = \mathcal{NP}$ .

- This theorem essentially states that if  $\mathcal{P} \cap \mathcal{NPC}$  is non-empty iff  $\mathcal{P} = \mathcal{NP}$ .

## $\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still):  $\mathcal{P} \neq \mathcal{NP}$ .

## $\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still):  $\mathcal{P} \neq \mathcal{NP}$ .

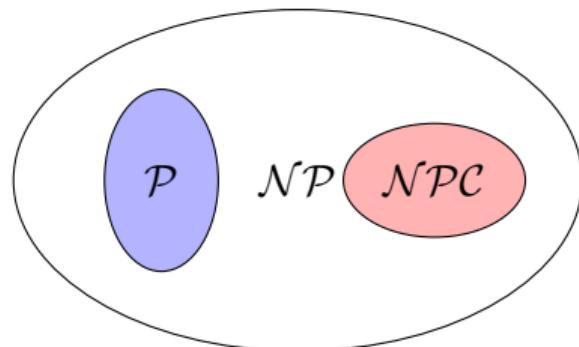


Figure:  $\mathcal{P} \neq \mathcal{NP}$

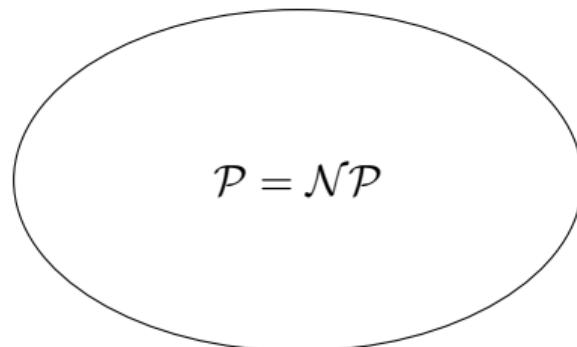


Figure:  $\mathcal{P} = \mathcal{NP}$

## $\mathcal{P}$ vs. $\mathcal{NP}$ revisited

Overwhelming consensus (still):  $\mathcal{P} \neq \mathcal{NP}$ .

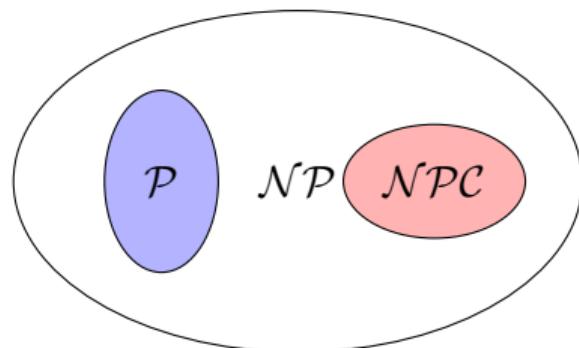


Figure:  $\mathcal{P} \neq \mathcal{NP}$

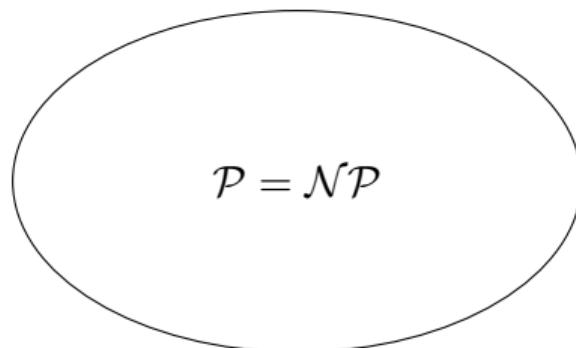
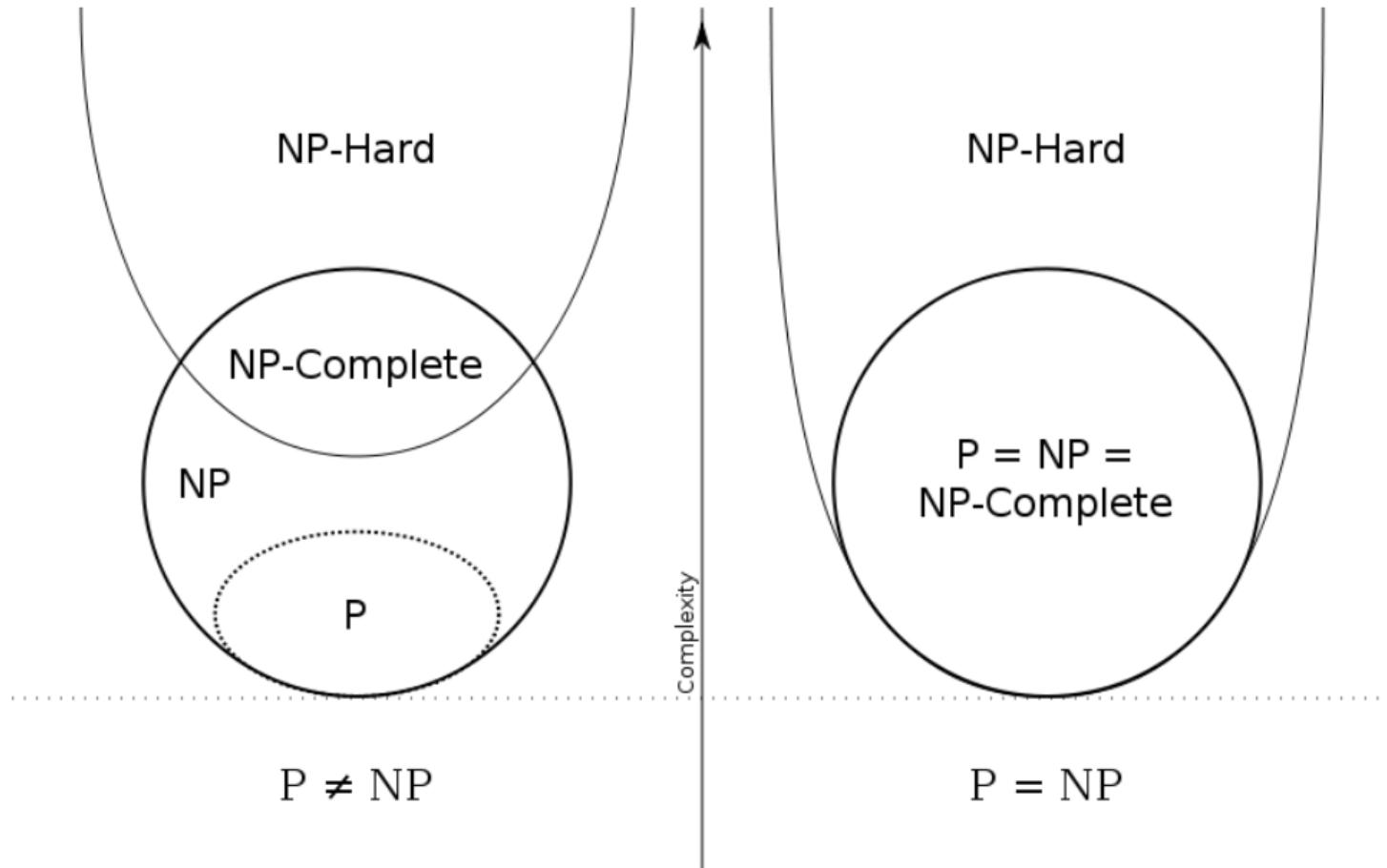


Figure:  $\mathcal{P} = \mathcal{NP}$

Why we believe  $\mathcal{P} \neq \mathcal{NP}$ ? Because some problems appear significantly harder.



## Cook-Levin Theorem

*Do there exist “natural”  $\text{NP}$ -complete problems?*

## Cook-Levin Theorem

*Do there exist “natural”  $\mathcal{NP}$ -complete problems?*



**Figure:** Stephen Cook & Leonid Levin

## Cook-Levin Theorem

*Do there exist “natural”  $\mathcal{NP}$ -complete problems?*



**Figure:** Stephen Cook & Leonid Levin

Cook-Levin theorem proves that  $SAT \in \mathcal{NPC}$ . Via Cook-Levin reduction, we can reduce any problem in  $\mathcal{NP}$  to SAT.

## Cook-Levin Theorem

*Do there exist “natural”  $\mathcal{NP}$ -complete problems?*



**Figure:** Stephen Cook & Leonid Levin

Cook-Levin theorem proves that  $SAT \in \mathcal{NPC}$ . Via Cook-Levin reduction, we can reduce any problem in  $\mathcal{NP}$  to SAT.

- Other examples of  $\mathcal{NP}$ -complete including graph 3-colorability, graph Hamiltonicity, and so on.

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Motivation for co-NP

Asymmetry of NP: We need short certificates only for yes instances.

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 1$$

$$x \notin L \iff \forall w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 0$$

## Motivation for co-NP

Asymmetry of NP: We need short certificates only for yes instances.

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 1$$
$$x \notin L \iff \forall w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 0$$

### SAT vs. UN-SAT

- Can prove a CNF formula is satisfiable by specifying an assignment.
- How could we prove that a formula is not satisfiable?

## Motivation for co-NP

Asymmetry of NP: We need short certificates only for yes instances.

$$x \in L \iff \exists w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 1$$
$$x \notin L \iff \forall w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } M(x, w) = 0$$

### SAT vs. UN-SAT

- Can prove a CNF formula is satisfiable by specifying an assignment.
- How could we prove that a formula is not satisfiable?

### HAM-CYCLE vs. NO-HAM-CYCLE

- Can prove a graph is Hamiltonian by specifying a path.
- How could we prove that a graph is not Hamiltonian?

Q: How to classify UN-SAT and NO-HAM-CYCLE?

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $\text{SAT} \equiv_p \text{UN-SAT}$ ,  $\text{HAM-CYCLE} \equiv_p \text{NO-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $SAT \equiv_p UN\text{-SAT}$ ,  $HAM\text{-CYCLE} \equiv_p NO\text{-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .
- $SAT \in \mathcal{NPC}$  and  $HAM\text{-CYCLE} \in \mathcal{NPC}$ , but neither UN-SAT nor NO-HAM-CYCLE are known to be in  $\mathcal{NP}$ .

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $SAT \equiv_p UN\text{-SAT}$ ,  $HAM\text{-CYCLE} \equiv_p NO\text{-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .
- $SAT \in \mathcal{NPC}$  and  $HAM\text{-CYCLE} \in \mathcal{NPC}$ , but neither UN-SAT nor NO-HAM-CYCLE are known to be in  $\mathcal{NP}$ .

Given a decision problem  $L \subseteq X$ , its complement  $\overline{L} \subseteq X$  is the same problem with the yes and no instances reversed.

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $\text{SAT} \equiv_p \text{UN-SAT}$ ,  $\text{HAM-CYCLE} \equiv_p \text{NO-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .
- $\text{SAT} \in \mathcal{NPC}$  and  $\text{HAM-CYCLE} \in \mathcal{NPC}$ , but neither UN-SAT nor NO-HAM-CYCLE are known to be in  $\mathcal{NP}$ .

Given a decision problem  $L \subseteq X$ , its complement  $\overline{L} \subseteq X$  is the same problem with the yes and no instances reversed.

### Example of Complement Problem

- $L = \{4, 6, 8, 9, 10, 12, 14, 15, \dots\}$ ,  $\overline{L} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \dots\}$

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $SAT \equiv_p UN\text{-SAT}$ ,  $HAM\text{-CYCLE} \equiv_p NO\text{-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .
- $SAT \in \mathcal{NPC}$  and  $HAM\text{-CYCLE} \in \mathcal{NPC}$ , but neither UN-SAT nor NO-HAM-CYCLE are known to be in  $\mathcal{NP}$ .

Given a decision problem  $L \subseteq X$ , its complement  $\overline{L} \subseteq X$  is the same problem with the yes and no instances reversed.

### Example of Complement Problem

- $L = \{4, 6, 8, 9, 10, 12, 14, 15, \dots\}$ ,  $\overline{L} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \dots\}$

### Definition 9 (co- $\mathcal{NP}$ )

Complements of decision problems in  $\mathcal{NP}$ .

Q: How to classify UN-SAT and NO-HAM-CYCLE?

- $SAT \equiv_p UN\text{-SAT}$ ,  $HAM\text{-CYCLE} \equiv_p NO\text{-HAM-CYCLE}$ , but neither SAT nor HAM-CYCLE are known to be in  $\mathcal{P}$ .
- $SAT \in \mathcal{NPC}$  and  $HAM\text{-CYCLE} \in \mathcal{NPC}$ , but neither UN-SAT nor NO-HAM-CYCLE are known to be in  $\mathcal{NP}$ .

Given a decision problem  $L \subseteq X$ , its complement  $\overline{L} \subseteq X$  is the same problem with the yes and no instances reversed.

### Example of Complement Problem

- $L = \{4, 6, 8, 9, 10, 12, 14, 15, \dots\}$ ,  $\overline{L} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \dots\}$

### Definition 9 (co- $\mathcal{NP}$ )

Complements of decision problems in  $\mathcal{NP}$ .

Examples: UN-SAT, NO-HAM-CYCLE, and PRIMES.

## $\mathcal{NP}$ vs. co- $\mathcal{NP}$

Fundamental open question: Does  $\mathcal{NP} = \text{co-}\mathcal{NP}$ ?

## $\mathcal{NP}$ vs. co- $\mathcal{NP}$

Fundamental open question: Does  $\mathcal{NP} = \text{co-}\mathcal{NP}$ ?

- Do yes instances have succinct certificates iff no instances do?

## $\mathcal{NP}$ vs. co- $\mathcal{NP}$

Fundamental open question: Does  $\mathcal{NP} = \text{co-}\mathcal{NP}$ ?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

## $\mathcal{NP}$ vs. co- $\mathcal{NP}$

Fundamental open question: Does  $\mathcal{NP} = \text{co-}\mathcal{NP}$ ?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

### Theorem 10

If  $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ , then  $\mathcal{P} \neq \mathcal{NP}$ .

## $\mathcal{NP}$ vs. co- $\mathcal{NP}$

Fundamental open question: Does  $\mathcal{NP} = \text{co-}\mathcal{NP}$ ?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

### Theorem 10

If  $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ , then  $\mathcal{P} \neq \mathcal{NP}$ .

Proof idea.

- $\mathcal{P}$  is closed under complementation.
- If  $\mathcal{P} = \mathcal{NP}$ , then  $\mathcal{NP}$  is closed under complementation. In other words,  $\mathcal{NP} = \text{co-}\mathcal{NP}$ .
- This is the contrapositive of the theorem.

$\mathcal{NP} \cap \text{co-}\mathcal{NP}$

Good characterization: [Edmonds 1965]  $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

## $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

Good characterization: [Edmonds 1965]  $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

If problem  $X \in \mathcal{NP} \cap \text{co-}\mathcal{NP}$ , then:

- for yes instance, there is a succinct certificate
- for no instance, there is a succinct disqualifier

## $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

Good characterization: [Edmonds 1965]  $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

If problem  $X \in \mathcal{NP} \cap \text{co-}\mathcal{NP}$ , then:

- for yes instance, there is a succinct certificate
- for no instance, there is a succinct disqualifier

Provides conceptual leverage for reasoning about a problem.

## Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Motivation of Randomized Algorithm

TM models deterministic algorithms.

TM does not seem to capture one aspect of reality — the ability to make random choices during computation

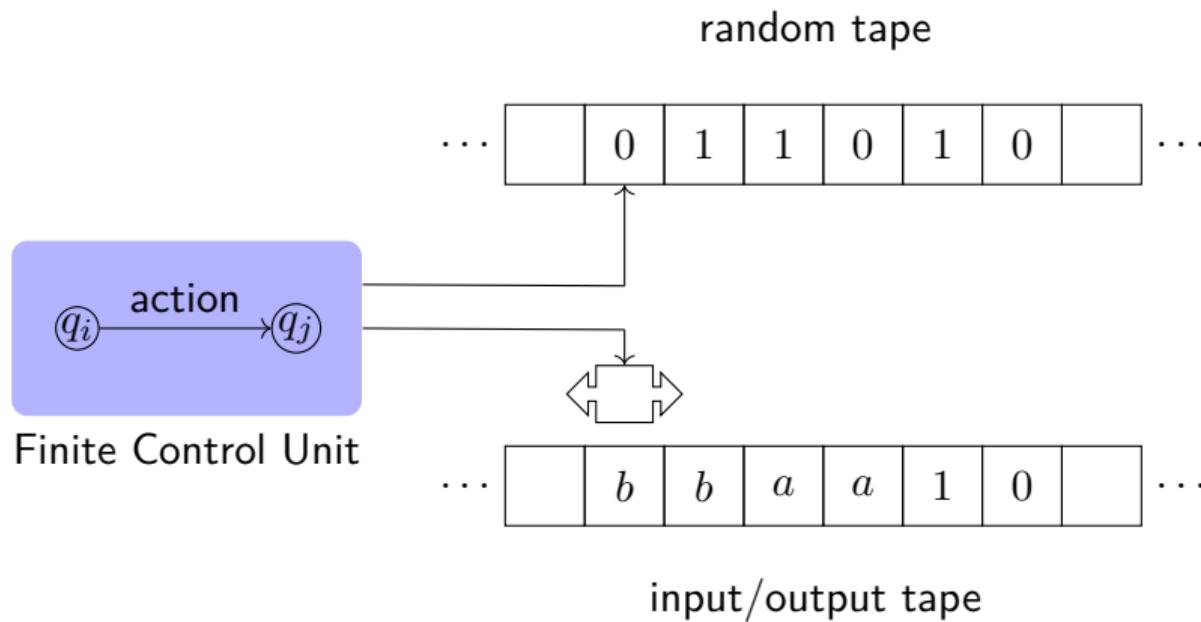
- Most programming languages provide a built-in RNG.

It makes sense to consider algorithms that can toss a coin, a.k.a. use a source of random bits. Such algorithms have been implicitly studied for a long time.

- estimate facts about a large sample by taking a small sample
- simulate real-world systems that are themselves probabilistic, such as nuclear fission and the stock market
- differential equations

## Probabilistic Turing Machine

Probabilistic Polynomial-time TM models probabilistic algorithm.



## PTM vs. NDTM

NDTM is a TM with two transition functions. PTM is syntactically similar.

The difference is in how we interpret the working of TM.

- In a PTM, each transition is taken with probability  $1/2$ , a computation that runs for time  $t$  gives rise  $2^t$  branches in the graph of all computations, each of which is taken with probability  $1/2^t$ .  $\Pr[M(x) = 1]$  is simply the *fraction* of branches that end with  $M$  outputting a 1.
- In a NDTM,  $M(x) = 1$  iff there exists a branch that outputs 1

On a conceptual level, PTM and NDTM are very different

- PTM like TM and unlike NDTM, is intended to model realistic computation devices.

## Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Bounded-Error Probabilistic Polynomial Time

### Definition 11 ( $\mathcal{BPP}$ Complexity)

$L \in \mathcal{BPP}$  iff there exists a probabilistic polynomial time TM  $M$  such that:

$$\forall x \in L : \Pr[M(x) = 1] \geq \frac{1}{2} + \epsilon$$

$$\forall x \notin L : \Pr[M(x) = 1] \leq \frac{1}{2} - \epsilon$$

**Intuition:**  $M$  is more likely correct than wrong.

## Bounded-Error Probabilistic Polynomial Time

### Definition 11 ( $\mathcal{BPP}$ Complexity)

$L \in \mathcal{BPP}$  iff there exists a probabilistic polynomial time TM  $M$  such that:

$$\forall x \in L : \Pr[M(x) = 1] \geq \frac{1}{2} + \epsilon$$

$$\forall x \notin L : \Pr[M(x) = 1] \leq \frac{1}{2} - \epsilon$$

### Key properties

- Bounded error:  $M$  must be noticeably more likely to be correct than incorrect, i.e., the error is bounded away from the random guess  $1/2$  by a noticeable gap.
- Error reduction: By repetition and then taking majority vote, the error can be reduced to  $2^{-n}$  with only polynomial overhead.
- Practical meaning:  $\mathcal{BPP}$  captures efficient randomized algorithms used in practice (e.g., primality testing before AKS).

**Intuition:**  $M$  is more likely correct than wrong.

## Error Reduction

In practice, an error probability  $1/2 - \epsilon$  (the gap is  $\epsilon$ ) might not be acceptable.

## Error Reduction

In practice, an error probability  $1/2 - \epsilon$  (the gap is  $\epsilon$ ) might not be acceptable.

However, the choice of gap  $\epsilon$  could be arbitrary and the set  $\mathcal{BPP}$  will be unchanged.

- It can be any constant between  $(0, 1/2)$ .
- It does not even have to be constant:  $\epsilon$  could be as small as  $n^{-c}$ , where  $c$  is any positive constant, and  $n$  is the length of input.

## Error Reduction

In practice, an error probability  $1/2 - \epsilon$  (the gap is  $\epsilon$ ) might not be acceptable.

However, the choice of gap  $\epsilon$  could be arbitrary and the set  $BPP$  will be unchanged.

- It can be any constant between  $(0, 1/2)$ .
- It does not even have to be constant:  $\epsilon$  could be as small as  $n^{-c}$ , where  $c$  is any positive constant, and  $n$  is the length of input.

The method is running  $M$  independently several times, then taking the majority vote.

**Chernoff Bound** guarantees the probability that the majority of the runs are wrong drops off exponentially.

**Chernoff Bound (Lower Tail):** Let  $\{X_i\}$  be independent 0-1 random variables such that  $\Pr[X_i = b] \geq 1/2 + \epsilon$ ,  $X = \sum_{i=1}^m X_i$ .

$$\Pr[X \neq b] \leq \frac{1}{e^{\epsilon^2 m / 2}}$$

Setting  $m \geq n^{2c} + 1$  ensures the error probability is negligible in  $n$ .

## Why Does Soundness Amplification Not Work?

In the ZKP literature, the error we care about is soundness error:

$$\epsilon = \Pr[\langle P, V \rangle(x) : x \notin L]$$

We can reduce  $\epsilon$  to  $\epsilon^n$  by:

- Run the protocol  $n$  times in sequential/parallel
- Accept iff all of the repetitions accept

Such method does not work for  $\mathcal{BPP}$ , cause in ZKP the completeness error is typically 0, while in  $\mathcal{BPP}$  the error is two-sided.

- The decision algorithm does not know the input instance belongs to YES or NO. It has to apply the same strategy.

## Probabilistic Polynomial Time

### Definition 12 ( $\mathcal{PP}$ Complexity)

$L \in \mathcal{PP}$  iff there exists a probabilistic polynomial time TM  $M$  such that:

$$\forall x \in L : \Pr[M(x) = 1] \geq 1/2$$

$$\forall x \notin L : \Pr[M(x) = 1] \leq 1/2$$

## Probabilistic Polynomial Time

### Definition 12 ( $\mathcal{PP}$ Complexity)

$L \in \mathcal{PP}$  iff there exists a probabilistic polynomial time TM  $M$  such that:

$$\forall x \in L : \Pr[M(x) = 1] \geq 1/2$$

$$\forall x \notin L : \Pr[M(x) = 1] \leq 1/2$$

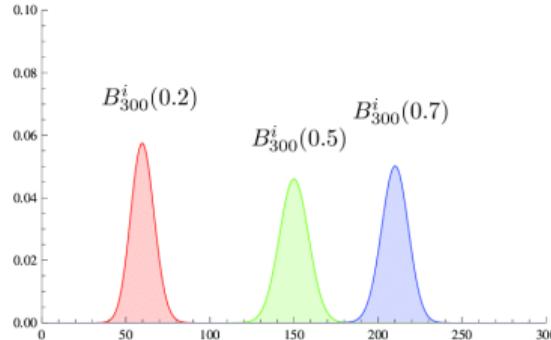
### Key properties

- Unbounded error: The acceptance probability may exceed 1/2 by an exponentially tiny amount.
- No error reduction: The gap away from 1/2 may be too small to reduce efficiently.
- Extremely powerful:  $\mathcal{PP}$  contains many hard classes.

**Intuition:**  $M$  is just barely more likely correct than wrong.

## Why Does Majority Vote Work?

Recall that  $X = X_1 + \dots + X_m$  is a Binomial distribution. When  $\epsilon$  is noticeable, after polynomial times of repetition, the two distributions induced by  $x \in L$  and  $x \notin L$  are largely detached. Otherwise, they are mingled together  $\rightsquigarrow$  hard to find a split line



This also explain for  $\mathcal{PP}$ , the majority vote does not work if  $\epsilon$  is negligibly close to 0. For example:

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{2} + \frac{1}{2^n}$$

$$x \notin L \Rightarrow \Pr[M(x) = 1] \leq \frac{1}{2} - \frac{1}{2^n}$$

### Containment

$$\mathcal{BPP} \subseteq \mathcal{PP}$$

- Any bounded-error algorithm is also an unbounded-error one.
  - The containment is believed to be strict.
- 

The difference between  $n^{-c}$  vs  $2^{-n}$  looks small numerically, but it is enormous computationally:

- $\mathcal{BPP}$  requires a noticeable probability advantage and models realistic randomized computation
- $\mathcal{PP}$  allows an infinitesimal advantage over guessing, making it far more powerful and closer to counting complexity than to efficient computation.

## About $\mathcal{BPP}$

$\mathcal{BPP}$  is one of the largest practical class of problems, since problems in  $\mathcal{BPP}$  have efficient probabilistic algorithms that can be run quickly on real modern machines.

## About $\mathcal{BPP}$

$\mathcal{BPP}$  is one of the largest practical class of problems, since problems in  $\mathcal{BPP}$  have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly,  $\mathcal{BPP} \supseteq \mathcal{P}$ , since a deterministic machine is a special case of a probabilistic machine.

## About $\mathcal{BPP}$

$\mathcal{BPP}$  is one of the largest practical class of problems, since problems in  $\mathcal{BPP}$  have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly,  $\mathcal{BPP} \supseteq \mathcal{P}$ , since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in  $\mathcal{BPP}$  but not known to be in  $\mathcal{P}$ . The number of such problems is decreasing, and it is conjectured that  $\mathcal{P} = \mathcal{BPP}$ .

## About $\mathcal{BPP}$

$\mathcal{BPP}$  is one of the largest practical class of problems, since problems in  $\mathcal{BPP}$  have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly,  $\mathcal{BPP} \supseteq \mathcal{P}$ , since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in  $\mathcal{BPP}$  but not known to be in  $\mathcal{P}$ . The number of such problems is decreasing, and it is conjectured that  $\mathcal{P} = \mathcal{BPP}$ .

For a long time, one of the most famous problems that was known to be in  $\mathcal{BPP}$  but not known to be in  $\mathcal{P}$  was the PRIME.

## About $\mathcal{BPP}$

$\mathcal{BPP}$  is one of the largest practical class of problems, since problems in  $\mathcal{BPP}$  have efficient probabilistic algorithms that can be run quickly on real modern machines.

- Clearly,  $\mathcal{BPP} \supseteq \mathcal{P}$ , since a deterministic machine is a special case of a probabilistic machine.
- Many problems were known to be in  $\mathcal{BPP}$  but not known to be in  $\mathcal{P}$ . The number of such problems is decreasing, and it is conjectured that  $\mathcal{P} = \mathcal{BPP}$ .

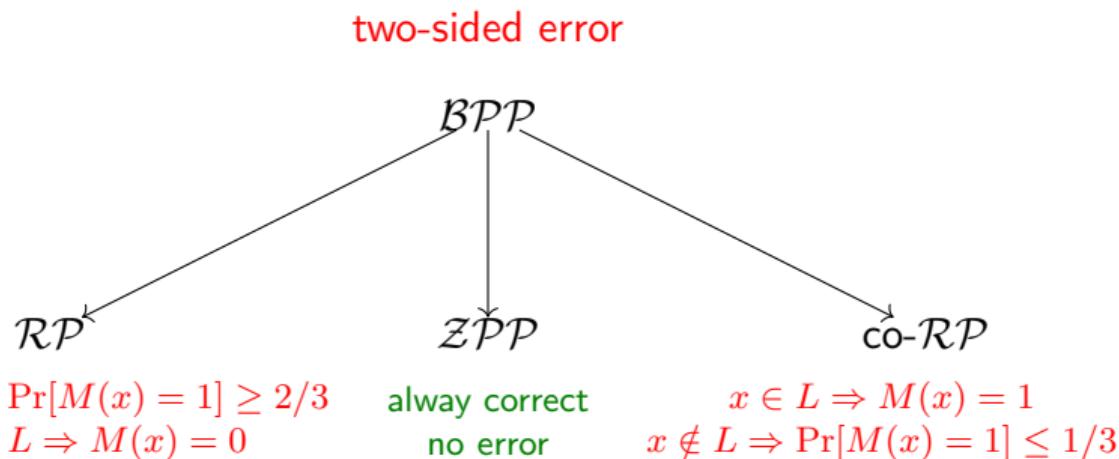
For a long time, one of the most famous problems that was known to be in  $\mathcal{BPP}$  but not known to be in  $\mathcal{P}$  was the PRIME.

[AKS04] gave a deterministic polynomial-time algorithm for PRIME, thus showing that it is in  $\mathcal{P}$ .

Gödel Prize and Fulkerson Prize

## One-sided and Zero-sided Error

$\mathcal{ZPP}$ : probabilistic polynomial-time TM always returns correct YES or NO answer, or halts with low probability, a.k.a. running time is polynomial **in expectation** for every input

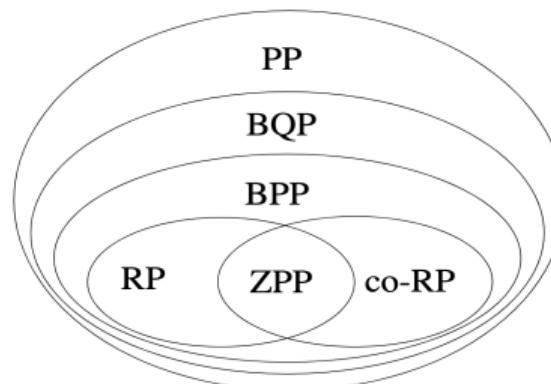


- $\mathcal{BPP}$ : Monte Carlo algorithms (probabilistic) likely to be correct in strict polynomial running time
- $\mathcal{ZPP}$ : Las Vegas algorithms (probabilistic) are always correct in expected polynomial running time

## $\mathcal{BPP}$ in Relation to Other Probabilistic Complexity Classes

$\mathcal{BQP}$  (bounded-error quantum polynomial time): the class of decision problems solvable by a quantum TM in polynomial time with bounded error

- It is the quantum analogue of  $\mathcal{BPP}$



## Limits of $\mathcal{BPP}$

Consensus:  $\mathcal{P} \subseteq \underline{\mathcal{ZPP}} = \mathcal{RP} \cap \text{co-}\mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{NP}$

$\mathcal{P} \subseteq \mathcal{BPP}$

- An important example of a problem in  $\mathcal{BPP}$  still not known to be in  $\mathcal{P}$  is **polynomial identity testing** — determining whether a polynomial is identically equal to the zero polynomial, when you have access to the value of the polynomial for any given input, but not to the coefficients.

$\mathcal{BPP} \subseteq \mathcal{NP}$

- Adleman's theorem:  $\mathcal{BPP} \subseteq P/\text{poly}$  (polynomial-size Boolean circuits)
- Karp-Levin theorem:  $\mathcal{NP} \subseteq P/\text{poly} \Rightarrow \text{PH} = \sum_2^P$

Thus,  $\mathcal{NP} \subseteq \mathcal{BPP}$  will imply collapse of PH, which is unlikely to be true. In other words,  $\nexists$  bounded-error probabilistic algorithms for  $\mathcal{NPC}$  problems.

## Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## PSPACE

$\mathcal{P}$ : Decision problems solvable by DTM in polynomial time.

PSPACE: Decision problems solvable by DTM in polynomial space.

### PSPACE Example

- Binary counter. Count from 0 to  $2^n - 1$  in binary.
- Algorithm. Use  $n$  bit odometer.

### Observation $\mathcal{P} \subseteq \text{PSPACE}$

- This is because poly-time algorithm can consume only polynomial space.

## Relation Between NP and PSPACE

Claim:  $\text{3-SAT} \in \text{PSPACE}$ .

Proof of claim

- Enumerate all  $2^n$  possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses.

Theorem:  $\mathcal{NP} \subseteq \text{PSPACE}$

- Consider arbitrary problem  $Y \in \mathcal{NP}$ .
- Since  $Y \leq_p \text{3-SAT}$ , there exists algorithm that solves  $Y$  in poly-time plus polynomial number of calls to 3-SAT black box.
- Thus,  $Y$  can be solved by DTM in poly-space.

It is easy to verify that  $\text{co-}\mathcal{NP} \subseteq \text{PSPACE}$ .

## The Merit of 3-SAT

Intuitively, with polynomial space DTM can simply enumerate all witness then check.  
Why we have resort to 3-SAT?

## The Merit of 3-SAT

Intuitively, with polynomial space DTM can simply enumerate all witness then check.  
Why we have resort to 3-SAT?

The reason is for some  $\mathcal{NP}$  language, we only know the asymptotic bound of  $|w|$ .

## The Merit of 3-SAT

Intuitively, with polynomial space DTM can simply enumerate all witness then check.  
Why we have resort to 3-SAT?

The reason is for some  $\mathcal{NP}$  language, we only know the asymptotic bound of  $|w|$ .

But for 3-SAT, given an instance  $x$ , we know the exact length of its witness.

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Decision vs. Search

So far, we only consider decision problems, i.e., membership problem

## Decision vs. Search

So far, we only consider decision problems, i.e., membership problem

More often, we want to find the solutions of given instances, i.e., the corresponding witness.

- Such kind of problems are *search* problem in nature.

## Decision vs. Search

So far, we only consider decision problems, i.e., membership problem

More often, we want to find the solutions of given instances, i.e., the corresponding witness.

- Such kind of problems are *search* problem in nature.

Q: *If decision problems are sufficiently expressive? Or as hard as its search version?*

## Decision vs. Search

So far, we only consider decision problems, i.e., membership problem

More often, we want to find the solutions of given instances, i.e., the corresponding witness.

- Such kind of problems are *search* problem in nature.

Q: *If decision problems are sufficiently expressive? Or as hard as its search version?*

A: For  $\mathcal{NP}$  languages. Yes!

## Example of 3-SAT

3-SAT (decision): decide if a given formula  $\psi$  is satisfiable

## Example of 3-SAT

3-SAT (decision): decide if a given formula  $\psi$  is satisfiable

3-SAT (search): if a given formula  $\psi$  is satisfiable, find one of its assignment

## Example of 3-SAT

3-SAT (decision): decide if a given formula  $\psi$  is satisfiable

3-SAT (search): if a given formula  $\psi$  is satisfiable, find one of its assignment

### Theorem 13

*If there exists polynomial-time algorithm for 3-SAT, then there exists a polynomial-time algorithm that finds its assignment.*

## Example of 3-SAT

3-SAT (decision): decide if a given formula  $\psi$  is satisfiable

3-SAT (search): if a given formula  $\psi$  is satisfiable, find one of its assignment

### Theorem 13

*If there exists polynomial-time algorithm for 3-SAT, then there exists a polynomial-time algorithm that finds its assignment.*

- define an  $\mathcal{NP}$  language

$$L = \{(\phi, r) : \exists w \text{ s.t. } \phi(w) = 1 \wedge w < r\}$$

- given black-box access for  $L$  (at most  $\log |W|$ ), one can use binary search to find  $w$ .

## Example of 3-SAT

3-SAT (decision): decide if a given formula  $\psi$  is satisfiable

3-SAT (search): if a given formula  $\psi$  is satisfiable, find one of its assignment

### Theorem 13

*If there exists polynomial-time algorithm for 3-SAT, then there exists a polynomial-time algorithm that finds its assignment.*

- define an  $\mathcal{NP}$  language

$$L = \{(\phi, r) : \exists w \text{ s.t. } \phi(w) = 1 \wedge w < r\}$$

- given black-box access for  $L$  (at most  $\log |W|$ ), one can use binary search to find  $w$ .

Conclusion: decision 3-SAT  $\approx_{\text{hard}}$  search 3-SAT

## The Case of Cryptographic Problem

Apply Cook-Levin reduction, all  $\mathcal{NP}$  languages enjoy the same property.

## The Case of Cryptographic Problem

Apply Cook-Levin reduction, all  $\mathcal{NP}$  languages enjoy the same property.

Q: How about cryptographic problems? For a search problem (e.g. factoring), can we transform it into a decision problem?

## The Case of Cryptographic Problem

Apply Cook-Levin reduction, all  $\mathcal{NP}$  languages enjoy the same property.

Q: How about cryptographic problems? For a search problem (e.g. factoring), can we transform it into a decision problem?

How about factoring? (find the unique factorization of  $N$ )

## The Case of Cryptographic Problem

Apply Cook-Levin reduction, all  $\mathcal{NP}$  languages enjoy the same property.

Q: How about cryptographic problems? For a search problem (e.g. factoring), can we transform it into a decision problem?

How about factoring? (find the unique factorization of  $N$ )

- To consider the computational complexity of factoring, we need to make a language version.
- Define  $\text{Factor} = \{(N, r) : \exists s \text{ s.t. } 1 < s < r \wedge s|N\}$ .  $(m, r) \in \text{FACTOR}$  iff  $r$  is greater than the least prime factor of  $m$ .

## The Case of Cryptographic Problem

Apply Cook-Levin reduction, all  $\mathcal{NP}$  languages enjoy the same property.

Q: How about cryptographic problems? For a search problem (e.g. factoring), can we transform it into a decision problem?

How about factoring? (find the unique factorization of  $N$ )

- To consider the computational complexity of factoring, we need to make a language version.
- Define  $\text{Factor} = \{(N, r) : \exists s \text{ s.t. } 1 < s < r \wedge s|N\}$ .  $(m, r) \in \text{FACTOR}$  iff  $r$  is greater than the least prime factor of  $m$ .
- Given a black-box for FACTOR one can use binary search to find a prime factor of  $m$ .

# Outline

1 Decision Problem

2 Deterministic Computation

3 Several Important Complexity Classes

- $\mathcal{P}$  vs.  $\mathcal{NP}$
- $\mathcal{NP}$ -complete
- co- $\mathcal{NP}$

4 Randomized Computation

- $\mathcal{BPP}$
- PSPACE

5 Decision vs. Search

6 Impact on Cryptography

## Languages Defined by Cryptographic Problems

Example based on DDH w.r.t.  $(\mathbb{G}, p, g_1, g_2)$

- $X = \mathbb{G} \times \mathbb{G}$ ;
- $L_{g_1, g_2} = \{(g_1^w, g_2^w)\}$  where  $w \in \mathbb{Z}_p$ ;

## Languages Defined by Cryptographic Problems

Example based on DDH w.r.t.  $(\mathbb{G}, p, g_1, g_2)$

- $X = \mathbb{G} \times \mathbb{G}$ ;
- $L_{g_1, g_2} = \{(g_1^w, g_2^w)\}$  where  $w \in \mathbb{Z}_p$ ;

Example based on PRG:  $\{0, 1\}^n \rightarrow \{0, 1\}^{2n}$

- $X = \{0, 1\}^{2n}$ ;
- $L = \{\text{PRG}(s) : s \in \{0, 1\}^n\}$ ;

## Languages Defined by Cryptographic Problems

Example based on DDH w.r.t.  $(\mathbb{G}, p, g_1, g_2)$

- $X = \mathbb{G} \times \mathbb{G}$ ;
- $L_{g_1, g_2} = \{(g_1^w, g_2^w)\}$  where  $w \in \mathbb{Z}_p$ ;

Example based on PRG:  $\{0, 1\}^n \rightarrow \{0, 1\}^{2n}$

- $X = \{0, 1\}^{2n}$ ;
- $L = \{\text{PRG}(s) : s \in \{0, 1\}^n\}$ ;

Both of them are in  $\mathcal{NP}$ , but neither in  $\mathcal{P}$  (or even  $\mathcal{BPP}$ ) nor  $\mathcal{NPC}$ .

## Languages Defined by Cryptographic Problems

Example based on DDH w.r.t.  $(\mathbb{G}, p, g_1, g_2)$

- $X = \mathbb{G} \times \mathbb{G}$ ;
- $L_{g_1, g_2} = \{(g_1^w, g_2^w)\}$  where  $w \in \mathbb{Z}_p$ ;

Example based on PRG:  $\{0, 1\}^n \rightarrow \{0, 1\}^{2n}$

- $X = \{0, 1\}^{2n}$ ;
- $L = \{\text{PRG}(s) : s \in \{0, 1\}^n\}$ ;

Both of them are in  $\mathcal{NP}$ , but neither in  $\mathcal{P}$  (or even  $\mathcal{BPP}$ ) nor  $\mathcal{NPC}$ .

Official answer: lie in  $\mathcal{NP}$ -intermediate =  $\mathcal{NP} \setminus \mathcal{P} \cup \mathcal{NPC}$

## Languages Defined by Cryptographic Problems

Example based on DDH w.r.t.  $(\mathbb{G}, p, g_1, g_2)$

- $X = \mathbb{G} \times \mathbb{G}$ ;
- $L_{g_1, g_2} = \{(g_1^w, g_2^w)\}$  where  $w \in \mathbb{Z}_p$ ;

Example based on PRG:  $\{0, 1\}^n \rightarrow \{0, 1\}^{2n}$

- $X = \{0, 1\}^{2n}$ ;
- $L = \{\text{PRG}(s) : s \in \{0, 1\}^n\}$ ;

Both of them are in  $\mathcal{NP}$ , but neither in  $\mathcal{P}$  (or even  $\mathcal{BPP}$ ) nor  $\mathcal{NPC}$ .

Official answer: lie in  $\mathcal{NP}$ -intermediate =  $\mathcal{NP} \setminus \mathcal{P} \cup \mathcal{NPC}$

Remark

Cryptographic problems assume average-case complexity (for  $x \xleftarrow{R} X$ ), while problems in computer science consider worst-case complexity.

## What Kind of Problems Does Cryptography Need?

$\mathcal{NP}$ -hardness is insufficient for cryptography

## What Kind of Problems Does Cryptography Need?

$\mathcal{NP}$ -hardness is insufficient for cryptography

- Cryptography assumes the existence of average-case intractable problems in  $\mathcal{NP}$ .

## What Kind of Problems Does Cryptography Need?

$\mathcal{NP}$ -hardness is insufficient for cryptography

- Cryptography assumes the existence of **average-case intractable** problems in  $\mathcal{NP}$ .
- $\mathcal{NP}$ -hardness only gives us information about complexity *in the limit*.

## What Kind of Problems Does Cryptography Need?

$\mathcal{NP}$ -hardness is insufficient for cryptography

- Cryptography assumes the existence of **average-case intractable** problems in  $\mathcal{NP}$ .
- $\mathcal{NP}$ -hardness only gives us information about complexity *in the limit*.
- $\mathcal{NP}$ -hardness considers worst-case time. Even if  $\mathcal{NPC}$  problems are hard in the worst-case ( $\mathcal{P} \neq \mathcal{NP}$ ), they still could be efficiently solvable in the average-case.

## What Kind of Problems Does Cryptography Need?

$\mathcal{NP}$ -hardness is insufficient for cryptography

- Cryptography assumes the existence of **average-case intractable** problems in  $\mathcal{NP}$ .
- $\mathcal{NP}$ -hardness only gives us information about complexity *in the limit*.
- $\mathcal{NP}$ -hardness considers worst-case time. Even if  $\mathcal{NPC}$  problems are hard in the worst-case ( $\mathcal{P} \neq \mathcal{NP}$ ), they still could be efficiently solvable in the average-case.

### Remark

Proving the existence of hard-on-average problems in  $\mathcal{NP}$  using the  $\mathcal{P} \neq \mathcal{NP}$  assumption is a major open problem.

## What Cryptography Expect?

worst-case = average-case

- worst-case assumption is weaker than average-case

## What Cryptography Expect?

worst-case = average-case

- worst-case assumption is weaker than average-case

secure against quantum attack

- some problems of practical interest (e.g., factoring, DL) are known to be in  $\mathcal{BQP}$ , but suspected to be outside  $\mathcal{P}$ .

## What Cryptography Expect?

worst-case = average-case

- worst-case assumption is weaker than average-case

secure against quantum attack

- some problems of practical interest (e.g., factoring, DL) are known to be in  $\mathcal{BQP}$ , but suspected to be outside  $\mathcal{P}$ .

Leading candidate: Lattice problems (enjoy both)

## Reference I

-  Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.  
Primes is in p.  
*Annals of Mathematics*, 160 (2):781–793, 2004.