

# Computational Complexity Lecture 09

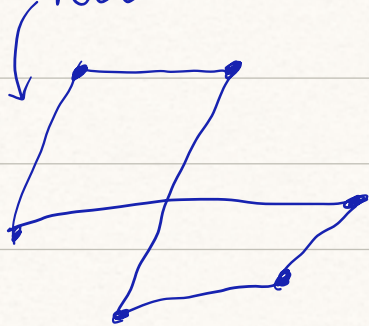
IP & Discrete Optimization Prob  $\Rightarrow$  Difficult (why)

Sec 1. Counting cost

we basically talk about the  
worst case

Q: How do we measure complexity? (An algorithm or a problem)

Ex 1 TSP  $\rightarrow$   $n$  Points



decidable  $\Leftrightarrow$  we can find the opt. soln  
in finite time

$\rightarrow$  all is  $(n-2)!$  tours

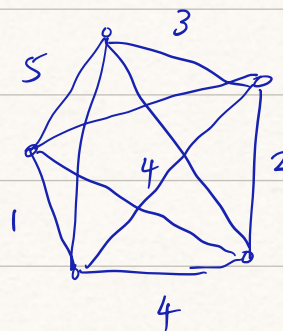
$\Downarrow$   
But this procedure is expensive!

Ex 2 Minimizing Spanning Tree

$\downarrow$   
decidable  
 $\Downarrow$

all # of possible spanning tree  
Foolish! is  $n^{n-2}$

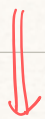
$\Downarrow$   
Expensive



a clever algorithm can  
terminate in time  $O(n^2)$

$\Downarrow$   
Fast!

Time taken  $\Rightarrow$  count cost



$\rightarrow$  depend on hardware  $\leftrightarrow$  not good

# of basic operations

+ , x

writing , reading

\* what we care about is  $\rightarrow$

growth rate of # operations

$\rightarrow$  How quickly it grows?

as a function of problem size

Defn :

when n is sufficiently BIG

①  $f(n) = O(g(n)) \Leftrightarrow f(n) \leq c g(n)$  for some  $c > 0$  EVENTUALLY

②  $f(n) = \Omega(g(n)) \Leftrightarrow f(n) \geq c g(n)$  for some  $c > 0$  eventually

③  $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$

Example

①  $f(n) = n$        $g(n) = \frac{1}{10} n^2$        $\Rightarrow f(n) = O(g(n))$       since  $f(n) \leq 1000 g(n)$

②  $f(n) = n+100$        $g(n) = \frac{1}{7} n+10$        $\Rightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$        $\Rightarrow f(n) = \Theta(g(n))$

③  $f(n) = n+3$        $g(n) = n^2+2n+3$

$\rightarrow f(n) = O(g(n))$

$3x^2+4 = O(x^2)$



dominant term

④  $f(n) = n^2 + 1000n$        $g(n) = n^2$

$\rightarrow f(n) = \Theta(g(n))$

⑤  $f(n) = \log n + n$        $g(n) = \log n$

$f(n) = \Omega(g(n))$

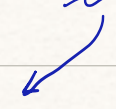
⑥  $f(n) = \log n + n$        $g(n) = (\log n)^2$



$$f(n) = \Omega(g(n))$$

Basic Idea: count the # of Basic operations as the 'size' of problem goes to  $\infty$ .

# of variables



Complexity of alg.

① Bubble sort  $\rightarrow O(n^2)$   $\left(\frac{n(n-1)}{2}\right)$

② Merge sort  $\rightarrow O(n \log n)$

③ Greedy alg. for minimum spanning tree  $\rightarrow O(|E| \log |V|)$


Q: What is an acceptable growth rate?

A:  $O(n^k) \rightarrow$  polynomial growth

Note:  $O(n \log n) \approx O(n^{1+\epsilon})$   $\rightarrow$  polynomial growth  $\rightarrow$  acceptable

$O(n^{\log n}) \rightarrow$  not polynomial growth  $\rightarrow$  unacceptable

[Rmk 1]: Decompose the whole task into several pieces  $\sim$  poly-time

  
still poly-time

[Rmk 2]:  $O(n^k)$  for large  $k$  are impractical. (quite often)

example: matrix multiplication  $\sim \underline{O(n^3)}$   
 $\rightarrow$  can be faster  $\underline{O(n^{2.3 \dots})}$

[Remark 3]: The Defn of poly-time alg. is independent on the computer hardware.

Q: What is the complexity of simplex algorithm?

A:  $O(e^n) \Rightarrow$  for the worst case  $\rightarrow$  for simple problem, it is very fast

Interior Point Method  $\sim$  poly-time alg.

$\downarrow$   
poly-time

---

## Section 2 : Recognition Problems



What constitutes a difficult problem?



there might be multiple ways to solve a single problem.

Defn: A recognition problem has binary response

$\hookrightarrow$  Answer is  $\begin{cases} \text{Yes} \\ \text{No} \end{cases}$

Example:

- ① Is soln of optimization problem less than or equal to 0?
- ② Is this integer a prime number?
- ③ Can we satisfy all clauses of problems?



Q: Are optimization problems recognition problems?

Yes!

→ Is there an feasible soln with objective value less than  $k$ ?

⇓  
Recognition Problem

↓  
use this repeatedly to solve an optimization problem!

---

### Section 3    The Class P

Defn:

We say a Recognition Problem in  $P$   
if it is solvable by a poly-time algorithm.

Example

- ① sort a list of numbers  $O(n^2)$
- ② check two finite sequences are equal  $O(n)$
- ③ Find the minimal spanning tree  $O(|E| \log |V|) \approx O(n^{1+\epsilon})$
- ④ Find the greatest common divisor between 2 integers
- ⑤ LP

{ Note: class  $P \rightsquigarrow$  easy problem

{ Note: defn of  $P$  goes beyond recognition problems.

---

## Section Class NP

Defn : A recognition Problem is in NP

if given a YES instance of such a problem,

there exists a certificate of size  $O(|Input|^k)$

and is verifiable in time  $O(|Input|^k)$

→ check it is 'Feasible' (proof)



in poly-time

Thm:

$$P \subseteq NP$$

★ NP stands for the problems that are solvable in poly-time using a non-deterministic Turing Machine

Open Problem :  $\exists Q \in NP$  but  $Q \notin P$  ?

---

## Section 5. Reduction

Hard Problems:



If problems of type A include problems of type B as a

special case ⇒

then problems of type A cannot be easier

than type B



Defn :

$P_1$  reduces  $P_2$  if you can show solve  $P_1$  by solving  $P_2$

poly-time plus poly-time computation.

$P_2$  is at least as hard as  $P_1$   $P_2$  can be harder than  $P_1$

$P_1$  transforms to  $P_2$  if there is a poly-time alg.

s.t. given  $p_1 \in P_1$ , outputs  $p_2 \in P_2$  such that

$$p_1 = \text{YES} \Leftrightarrow p_2 = \text{YES}$$

Example [3-SAT]  $\rightarrow$  Reduction



YES

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



takes poly-time

map: True  $\rightarrow 1$   
False  $\rightarrow 0$

convert

$$(x_1 \vee x_2 \vee \neg x_3)$$

$\rightarrow$

$$z_1 + z_2 + (1 - z_3) \geq 1$$

,  $z_i \in \{0, 1\}$

$$(x_2 \vee x_4 \vee \neg x_5)$$

$\rightarrow$

$$z_2 + z_4 + (1 - z_5) \geq 1$$

;

BIP

equivalent



Is there a feasible soln?

YES

Defn : the Problem  $P$  is NP-hard if all problems in NP reduce to  $P$

$\Rightarrow$  Corollary : The  $P$  is at least as hard (or harder) than any problem in NP

Defn:  $P$  is NP-complete if  $P$  is NP-hard and is in NP

$\Rightarrow$  Corollary:  $P$  is the hardest in NP

Note: 3-SAT reduces to the class of BIP.

$\Downarrow$

BIPs are NP-hard.  $\Rightarrow$  MILPs are also NP-hard.

$\Downarrow$

Actually are NP-complete.

Q: How do we go about showing that a problem is NP-hard?

$\downarrow$

A: Examine all NP-complete problems

$\downarrow$

show that there is a problem reduces to  
your current problem.

Q: How do we go about showing NP-complete?

A: ①  $P$  is NP-hard

②  $P$  is in NP



Q: If  $P_1$  reduces to  $P_2$  and  $P_2$  reduces to  $P_1$ , how do we make sense of this relation?

A: equally hard in Computational Cost

---

## Section 6 Closing Remarks

①  $P = NP$ ?

In fact, the world operates on the assumption that these two are different.

↳ (e.g. cryptographic hardness)

② Hardness is about the worst case scenarios.

↳ On average cases, it might be very easy to solve.

③ IPs are NP-complete.

⇓

we don't expect poly-time alg for solving

general IP.

④ It is possible to develop poly-time approximation algorithm that gives you solutions that are quite close.

Exam:

1. What is  $P$ ,  $NP$ ?

2. Show that a problem is in  $NP$ .

- What is the certificate

- Why it is a certificate

- Why it is poly-size and verifiable in poly-time

3. Perform a reduction.

- Give you a  $NP$ -complete Problem, reduce this to some current problems.

↓

You must show how  $P_1$  is a special case

of  $P_2$

hardone

- \* Show how  $P_1$  can reformulate as a special case of  $P_2$
- \* Yes in  $P_1 \Leftrightarrow$  Yes in  $P_2$
- \* process of converting from  $P_1$  to  $P_2$  is poly-time