P, NP, NP-Complete and NP-Hard

Introduction (1/1)

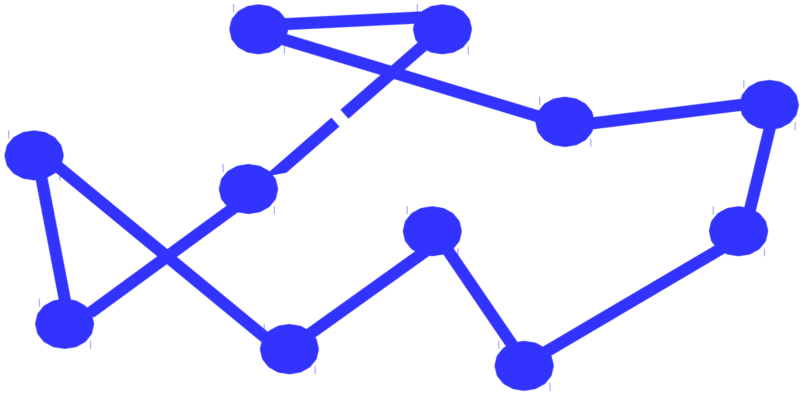
- Some Algorithms we've seen in this class
  – Sorting – O(N log N)
  – Searching – O(log N)
  – Shortest Path Finding – O(N^2)
  • However, some problems only have
  – Exponential Time Algorithm O(2^N)
  - 0/1 Knapsack problem O(2^N)
  - Travelling Salesman Problem O(2^N)
    – So What?

Introduction (2/2)

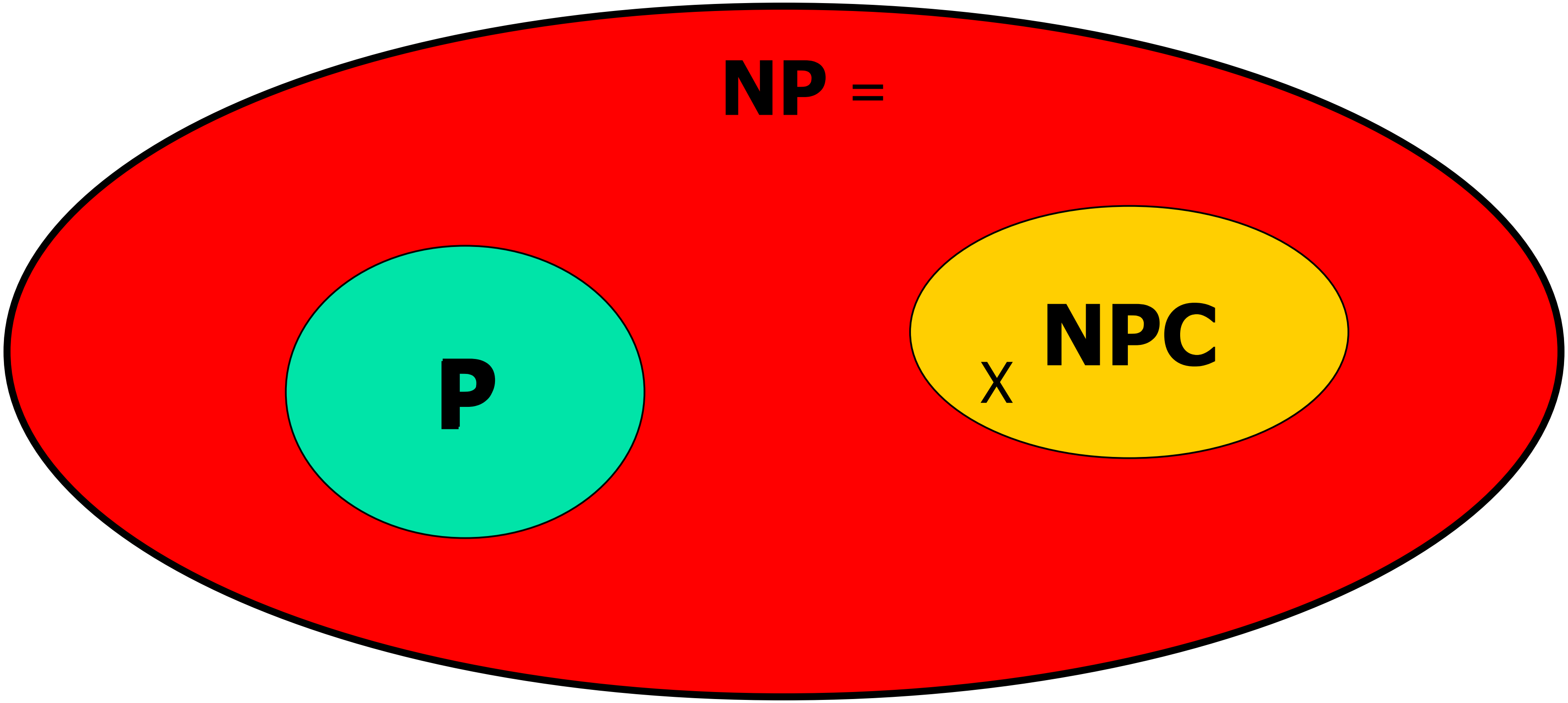
N	10	20	30	40	50	60
O(N)	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
$O(N^2)$	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
$O(N^3)$	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
$O(N^5)$	1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
$O(2^N)$	.001 second	1.0 second	17.9 minutes	**12.7 days**	**35.7 years**	**366 centuries**
[illegible]	.059 second	58 minutes	6.5 years	**3855 centuries**	**$2*10^8$ centuries**	**10^{13} centuries**

Motivation

- Traveling Salesman Problem (n = 1000)



- Compute 1000!
  - Even **Electron** in the Universe is a **Super Computer**,
  - And they work for the **Estimated Life of the Universe,**
  - **WE CANNOT SOLVE THIS PROBLEM!!!**
  - **This kind of problems are NP-Complete.**

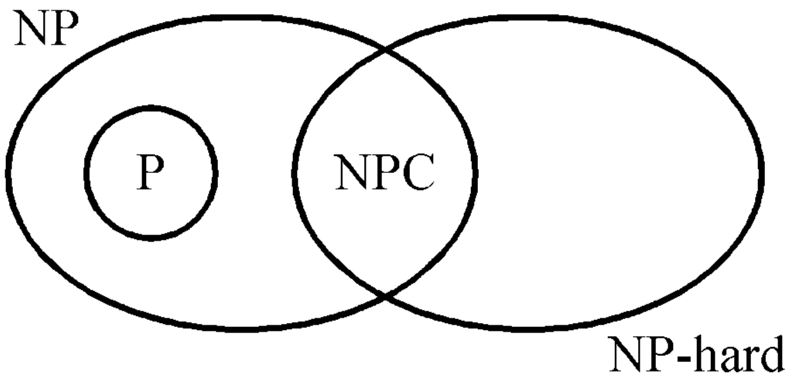


$P \stackrel{?}{=} NP$

NP: Non-deterministic Polynomial

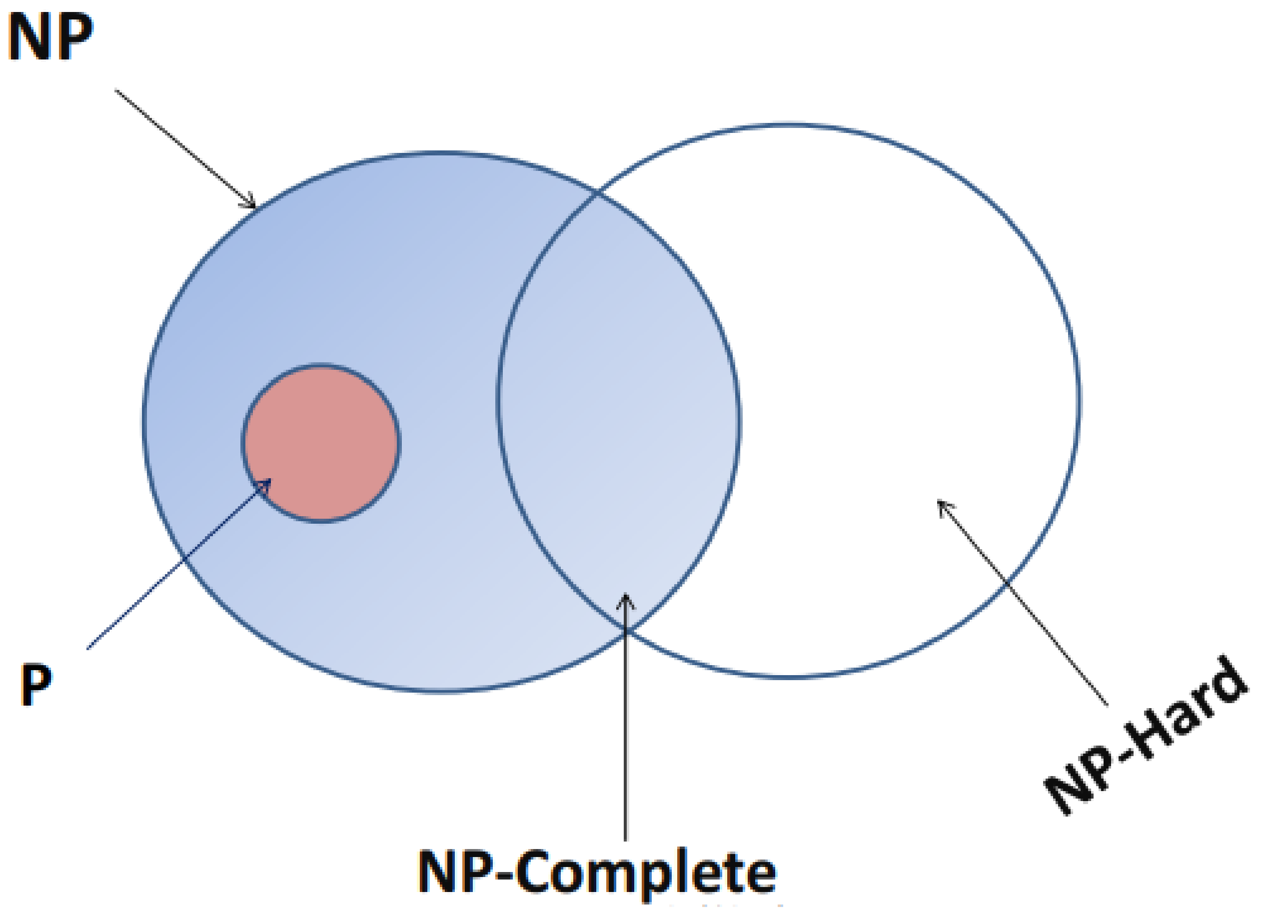
P: Polynomial

NPC: Non-deterministic Polynomial Complete



- **P**: the class of problems which can be solved by a deterministic polynomial algorithm.
- **NP** : the class of decision problem which can be solved by a non-deterministic polynomial algorithm.
- **NP-hard**: the class of problems to which every NP problem reduces.
- **NP-complete (NPC)**: the class of problems which are NP-hard and belong to NP.

Relationship among P, NP, NP-Complete and NP-Hard



Decision problems

- The solution is simply "Yes" or "No".
- Optimization problem : more difficult
  Decision problem
- E.g. the traveling salesperson problem
  - Optimization version:
    Find the shortest tour
  - Decision version:
    Is there a tour whose total length is less than or equal to a constant C ?

Nondeterministic algorithms

- **A nondeterministic algorithm is an algorithm consisting of two phases: guessing and checking.**
- **Furthermore, it is assumed that a nondeterministic algorithm always makes a correct guessing.**

Nondeterministic algorithms

- **They do not exist and they would never exist in reality.**
- **They are useful only because they will help us define a class of problems: NP problems**

NP algorithm

- If the checking stage of a nondeterministic algorithm is of polynomial time-complexity, then this algorithm is called an NP (nondeterministic polynomial) algorithm.

NP problem

- If a decision problem can be solved by a NP algorithm, this problem is called an NP (nondeterministic polynomial) problem.
- NP problems : (must be decision problems)

To express Nondeterministic Algorithm

- Choice(S) : arbitrarily chooses one of the elements in set S
- Failure : an unsuccessful completion
- Success : a successful completion

Nondeterministic searching Algorithm :

$j \leftarrow$ choice(1 : n) /* guess

if A(j) = x then success /* check

else failure

- A nondeterministic algorithm terminates unsuccessfully iff there exist no set of choices leading to a success signal.
- The time required for choice(1 : n) is O(1).
- A deterministic interpretation of a non-deterministic algorithm can be made by allowing unbounded parallelism in computation.

Problem Reduction

- Problem A reduces to problem B ($A \propto B$)
  - iff A can be solved by using any algorithm which solves B.
  - If $A \propto B$, B is more difficult (B is at least as hard as A)

$$
\begin{tikzcd}
\text{instance of A} \arrow[r, "\text{transformation}", "T(tr_1)"'] \arrow[d, "T(A)"'] & \text{instance of B} \arrow[d, "\text{solver of B}", "T(B)"'] \\
\text{answer of A} & \text{answer of B} \arrow[l, "\text{transformation}"', "T(tr_2)"]
\end{tikzcd}
$$

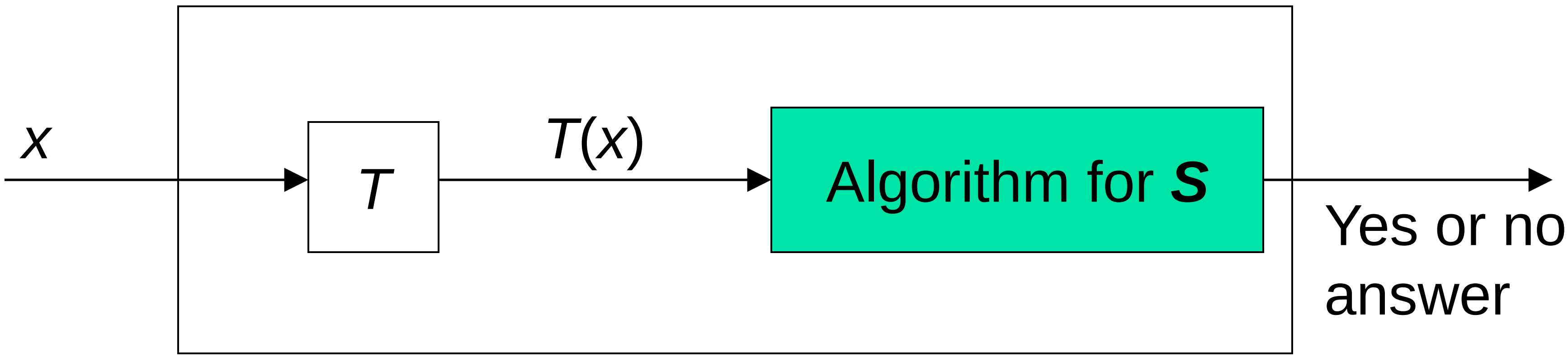
- Note: $T(tr_1) + T(tr_2) < T(B)$
- $T(A) \leq T(tr_1) + T(tr_2) + T(B) \sim O(T(B))$

Polynomial-time Reductions

- We want to solve a problem ***R***; we already have an algorithm for ***S***
- We have a transformation function *T*
  - Correct answer for ***R*** on *x* is "yes", iff the correct answer for ***S*** on *T*(*x*) is "yes"
- Problem ***R*** is *polynomially reducible* to ***S*** if such a transformation *T* can be computed in polynomial time
- The point of reducibility: ***S*** is at least as hard to solve as ***R***

Polynomial-time Reductions

- We use *reductions* (or *transformations)* to prove that a problem is **NP**-complete



Algorithm for ***R***

- x is an input for ***R***; $T(x)$ is an input for ***S***
- (R $\propto$ S)

NPC and NP-hard

- A problem A is NP-hard if every NP problem reduces to A.
- A problem A is NP-complete (NPC) if $A \in NP$ and every NP problem reduces to A.
  - Or we can say a problem A is NPC if $A \in NP$ and A is NP-hard.

NP-Completeness

- "***NP**-complete problems*": the hardest problems in **NP**
- Interesting property
  - If any *one* **NP**-complete problem can be solved in polynomial time, then *every* problem in **NP** can also be solved similarly (i.e., P=NP)
- Many believe **P≠NP**

Importance of **NP**-Completeness

- **NP**-complete problems: considered "intractable"
- Important for algorithm designers & engineers
- Suppose you have a problem to solve
  - Your colleagues have spent a lot of time to solve it exactly but in vain
  - See whether you can prove that it is **NP**-complete
  - If yes, then spend your time developing an *approximation* (*heuristic*) *algorithm*
- Many natural problems can be **NP**-complete

Relationship Between **NP** and **P**

- It is not known whether **P=NP** or whether **P** is a proper subset of **NP**
- It is believed **NP** is much larger than **P**
  - But no problem in **NP** has been proved as not in **P**
  - No known deterministic algorithms that are polynomially bounded for many problems in **NP**
  - So, “does **P** = **NP**?” is still an open question!

SAT is NP-complete

- Every NP problem can be solved by an NP algorithm
- Every NP algorithm can be transformed in polynomial time to an SAT problem (a Boolean formula C)
- Such that the SAT problem is satisfiable iff the answer for the original NP problem is "yes"
- That is, every NP problem $\propto$ SAT
- SAT is NP-complete

- Definition of the satisfiability problem: Given a Boolean formula, determine whether this formula is satisfiable or not.

- A literal: x_i or $-x_i$
- A clause: $x_1 \vee x_2 \vee -x_3 \equiv C_i$
- A formula: conjunctive normal form

$$C_1 \& C_2 \& \ldots \& C_m$$

The Satisfiability Problem

- The satisfiability problem
  - A logical formula:

$$x_1 \lor x_2 \lor x_3$$
$$\& - x_1$$
$$\& - x_2$$

the assignment :

$$x_1 \leftarrow F, \; x_2 \leftarrow F, \; x_3 \leftarrow T$$

will make the above formula true.

$(-x_1, -x_2, x_3)$ represents $x_1 \leftarrow F, \; x_2 \leftarrow F, \; x_3 \leftarrow T$

- If there is at least one assignment which satisfies a formula, then we say that this formula is satisfiable; otherwise, it is unsatisfiable.
- An unsatisfiable formula:

$$
\begin{aligned}
& x_1 \vee x_2 \\
\& \; & x_1 \vee -x_2 \\
\& \; & -x_1 \vee x_2 \\
\& \; & -x_1 \vee -x_2
\end{aligned}
$$

Traveling salesperson problem

- Given: A set of n planar points

  Find: A closed tour which includes all points exactly once such that its total length is minimized.
- This problem is NP-complete.

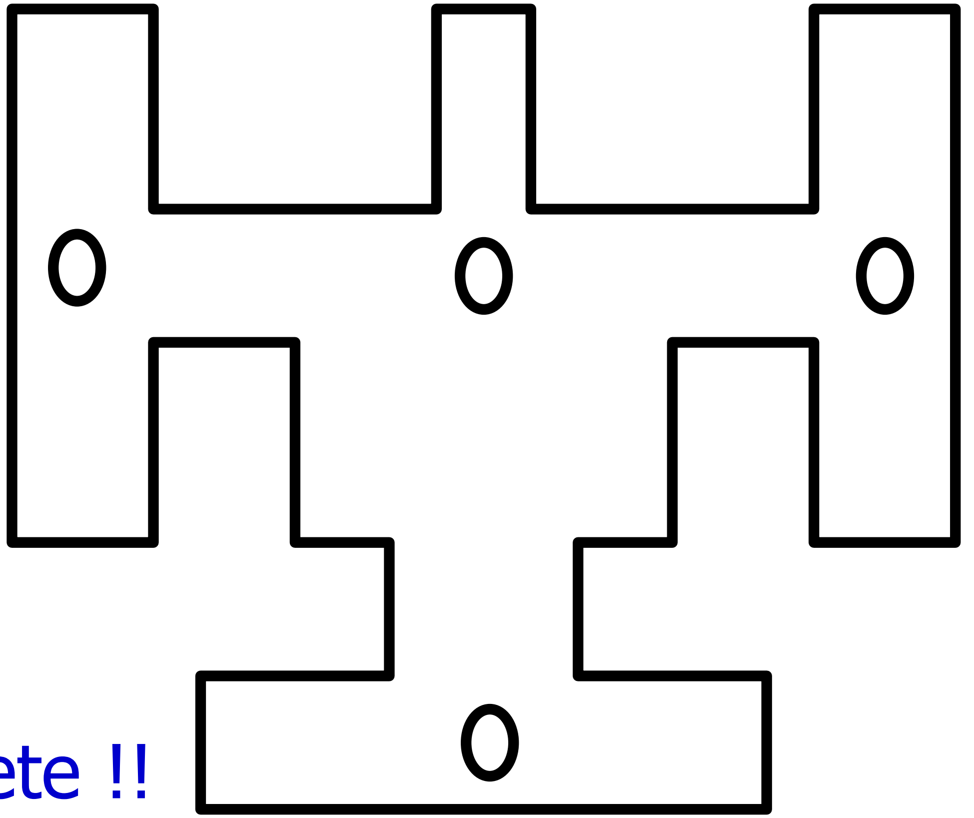
Partition problem

- Given: A set of positive integers S

  Find: S_1 and S_2 such that $S_1 \cap S_2 = \varnothing$, $S_1 \cup S_2 = S$, $\sum_{i \in S1} i = \sum_{i \in S2} i$

  (partition into S_1 and S_2 such that the sum of S_1 is equal to S_2)
- e.g. S={1, 7, 10, 9, 5, 8, 3, 13}
  - S_1={1, 10, 9, 8}
  - S_2={7, 5, 3, 13}
- This problem is NP-complete.

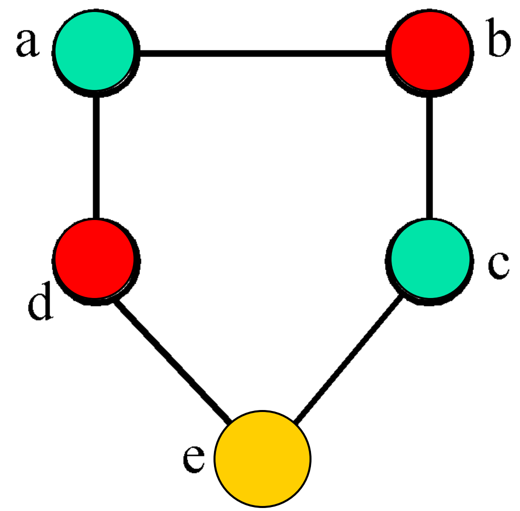
Art gallery problem



NP-complete !!

Chromatic Number Decision Problem (CN)

- **Def**: A coloring of a graph G = (V, E) is a function f: V $\rightarrow$ { 1, 2, 3,…, k } such that if (u, v) $\in$ E, then f(u)$\neq$f(v). The CN problem is to determine if G has a coloring for k.
- E.g.



3-colorable
f(a)=1, f(b)=2, f(c)=1
f(d)=2, f(e)=3

<Theorem> Satisfiability with at most 3 literals per clause (SATY) $\propto$ CN.