

# COSE215: Theory of Computation

## Lecture 20 — P, NP, and NP-Complete Problems

Hakjoo Oh  
2018 Spring

# Contents<sup>1</sup>

- $\mathcal{P}$  and  $\mathcal{NP}$
- Polynomial-time reductions
- NP-complete problems

---

<sup>1</sup>The slides are partly based on Siddhartha Sen's slides (“P, NP, and NP-Completeness”)

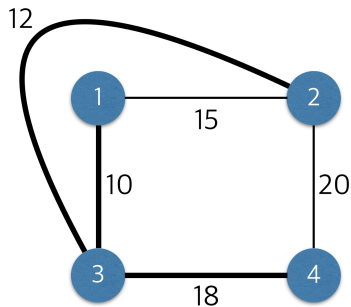
## Problems Solvable in Polynomial Time ( $\mathcal{P}$ )

- A Turing machine  $M$  is said to be of time complexity  $T(n)$  if whenever  $M$  is given an input  $w$  of length  $n$ ,  $M$  halts after making at most  $T(n)$  moves, regardless of whether or not  $M$  accepts.
  - ▶ E.g.,  $T(n) = 5n^2$ ,  $T(n) = 3^n + 5n^4$
- Polynomial time:  $T(n) = a_0n^k + a_1n^{k-1} + \dots + a_kn + a_{k+1}$
- We say a language  $L$  is in class  $\mathcal{P}$  if there is some polynomial  $T(n)$  such that  $L = L(M)$  for some deterministic TM  $M$  of time complexity  $T(n)$ .
- Problems solvable in polynomial time are called *tractable*.

## Example: Kruskal's Algorithm

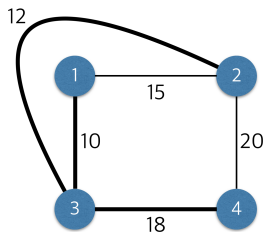
A greedy algorithm for finding a minimum-weight spanning tree for a weighted graph.

- a spanning tree: a subset of the edges such that all nodes are connected through these edges
- a minimum-weight spanning tree: a spanning tree with the least total weight



## Example: Kruskal's Algorithm

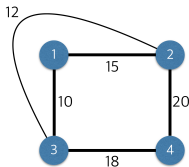
- Consider the edge (1,3) with the lowest weight (10). Because nodes 1 and 3 are not contained in  $T$  at the same time, include the edge in  $T$ .
- Consider the next edge in order of weights: (2,3). Since 2 and 3 are not in  $T$  at the same time, include (2,3) in  $T$ .
- Consider the next edge: (1,2). Nodes 1 and 2 are in  $T$ . Reject (1,2).
- Consider the next edge (3,4) and include it in  $T$ .
- We have three edges for the spanning tree of a 4-node graph, so stop.



The algorithm takes  $O(m + e \log e)$  steps ( $O(n^2)$  for multitape TM).

# Nondeterministic Polynomial Time ( $\mathcal{NP}$ )

- We say a language  $L$  is in the class  $\mathcal{NP}$  (nondeterministic polynomial) if there is a nondeterministic TM  $M$  and a polynomial time complexity  $T(n)$  such that  $L = L(M)$ , and when  $M$  is given an input of length  $n$ , there are no sequences of more than  $T(n)$  moves of  $M$ .
- Example: TSP (Travelling Salesman Problem)
  - ▶ finding a *hamiltonian cycle* (i.e., a cycle that contains all nodes and each node exactly once) with minimum cost: e.g.,



- ▶ To solve TSP, we need to try an exponential number of cycles and compute their total weight. Thus, TSP may not be in  $\mathcal{P}$ . TSP is in  $\mathcal{NP}$  because NTM can guess an exponential number of possible solutions and checking a hamiltonian cycle can be done in polynomial time.

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

One of the deepest open problems.

- In words: everything that can be done in polynomial time by an NTM can in fact be done by a DTM in polynomial time?
- $\mathcal{P} \subseteq \mathcal{NP}$  because every deterministic TM is a nondeterministic TM.
- $\mathcal{P} \supseteq \mathcal{NP}$ ? Probably not. It appears that  $\mathcal{NP}$  contains many problems not in  $\mathcal{P}$ . However, no one proved it.

## Implications of $\mathcal{P} = \mathcal{NP}$

*If  $P=NP$ , then the world would be a profoundly different place than we usually assume it to be. There would be no special value in “creative leaps,” no fundamental gap between solving a problem and recognizing the solution once it’s found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett.*

— Scott Aaronson



# NP-Complete Problems

- NP-complete problems are the hardest problems in the NP class.
- If any NP-complete problem can be solved in polynomial time, then all problems in NP are solvable in polynomial time.
- How to compare easiness/hardness of problems?

# Problem Solving by Reduction

- $L_1$ : the language (problem) to solve
- $L_2$ : the problem for which we have an algorithm to solve
- Solve  $L_1$  by reducing  $L_1$  to  $L_2$  ( $L_1 \leq L_2$ ) via function  $f$ :
  - 1 Convert input  $x$  of  $L_1$  to instance  $f(x)$  of  $L_2$ 
    - ★  $x \in L_1 \iff f(x) \in L_2$
  - 2 Apply the algorithm for  $L_2$  to  $f(x)$
- Running time = time to compute  $f$  + time to apply algorithm for  $L_2$
- We write  $L_1 \leq_P L_2$  if  $f(x)$  is computable in polynomial time

## Reductions show easiness/hardness

- To show  $L_1$  is easy, reduce it to something we know is easy
  - ▶  $L_1 \leq \text{easy}$
  - ▶ Use algorithm for easy language to decide  $L_1$
- To show  $L_1$  is hard, reduce something we know is hard to it (e.g., NP-complete problem)
  - ▶  $\text{hard} \leq L_1$
  - ▶ If  $L_1$  was easy, *hard* would be easy too

# NP-Complete Problems

We say  $L$  is NP-complete if

- 1  $L$  is in  $\mathcal{NP}$
- 2 For every language  $L'$  in  $\mathcal{NP}$ , there is a polynomial time reduction of  $L'$  to  $L$  (i.e.,  $L' \leq_P L$ )

# The Boolean Satisfiability Problem

Determine if the given boolean formula can be true.

- $x \wedge \neg x$
- $x \wedge \neg(y \vee z)$

The *first* problem proven to be NP-complete.

## Theorem (Cook-Levin)

*SAT is NP-complete.*

We need to show that

- 1 SAT is NP, and
- 2 for every L in NP, there is a polynomial-time reduction of L to SAT.

Many problems in artificial intelligence, automatic theorem proving, circuit design, etc reduce to SAT/SMT problems.

- E.g., see Z3 SMT solver (<https://github.com/Z3Prover/z3>)

# Summary

- Undecidable
- Decidable
  - ▶  $\mathcal{P}$
  - ▶  $\mathcal{NP}$
  - ▶ NP-complete