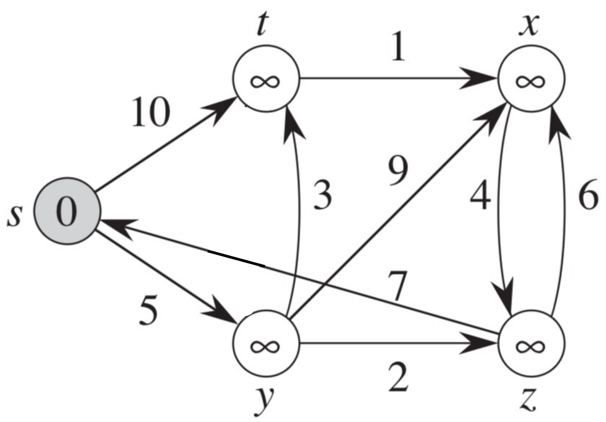
CS208
Ch. 8: NP-Completeness

Yang Xu, Ph.D.

Email: xuyang@sustech.edu.cn

Easy vs. hard problems

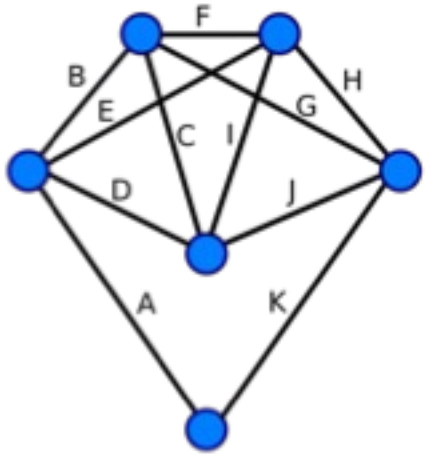
- **Shortest vs. longest simple paths:**
- *shortest* paths from a single source in a graph $G = (V, E)$ in $O(VE)$ time.
- However, finding a *longest* simple path between two vertices is difficult.
- Merely determining whether G contains a simple path with *at least* a given number of edges is hard.



(a)

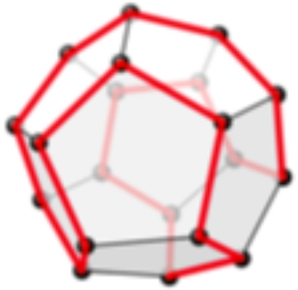
Easy vs. hard problems

- **Euler tour vs. Hamiltonian cycle**:
- A Euler tour (or, Eulerian path) of a graph is a cycle that traverses each *edge* exactly once.



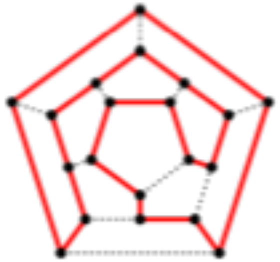
- We can determine whether a graph has a Euler tour in $O(E)$ time

- Determine whether a graph has a Hamiltonian cycle is **hard**



A polyhedron and its Hamiltonian cycle

- A Hamiltonian cycle of a graph is a simple cycle that visits each *vertex* exactly once.



Easy vs. hard problems

(合取范式)

- Conjunctive normal form (CNF): **2-CNF satisfiability vs. 3-CNF satisfiability**
- A Boolean formula contains variables whose values are 0 or 1;
- Boolean connectives: ∧ (AND), ∨ (OR), and ¬ (NOT);
- A Boolean formula is *satisfiable* if there exists some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1.

- A Boolean formula is in ***k-conjunctive normal form***, or ***k-CNF***, if it is the AND of clauses of ORs of exactly *k* variables or their negations.
- For example, the Boolean formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$ is in 2-CNF.
- It has the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 1$

- A 2-CNF satisfiability problem can be solved in polynomial time.
- However, a 3-CNF satisfiability problem **cannot**.
- E.g., $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$.

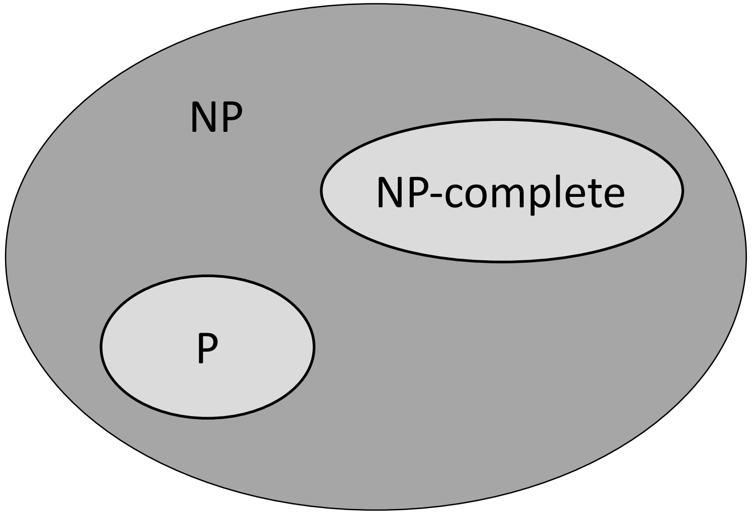
Three classes of problems

- Class **P**: Those problems that are ***solvable*** in **p**olynomial time.
  - They can be solved in $O(n^k)$ time, for some constant k, where n is the input size.
- Class **NP**: Those problems that are "***verifiable***" in **p**olynomial time.
  - If we are somehow given a solution to the problem, then we could verify if it is correct in polynomial time.
- Class **NP-complete**: If a problem is in NP, and is as "hard" as any problem in NP (*N*ondeterministic *P*olynomial-time)
  - The term "as hard as" can be formally defined.
  - NP-complete problems are those "hardest" among NP problems.
- The status of this class is unknown:
- **No** polynomial-time algorithm has yet been discovered for an NP-complete problem, **nor** has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them.

Intuitive relationship between P and NP

- Class **P** consists of problems that can be *solved* quickly.
- Class **NP** consists of problems that can be *verified* quickly.

- It is often more difficult to solve a problem from scratch than to verify a clearly presented solution.
- Therefore, it makes sense that P $\neq$ NP, i.e., there exists problems that are “NP-complete”.



All the problems so far …

- Are in the class **P**.

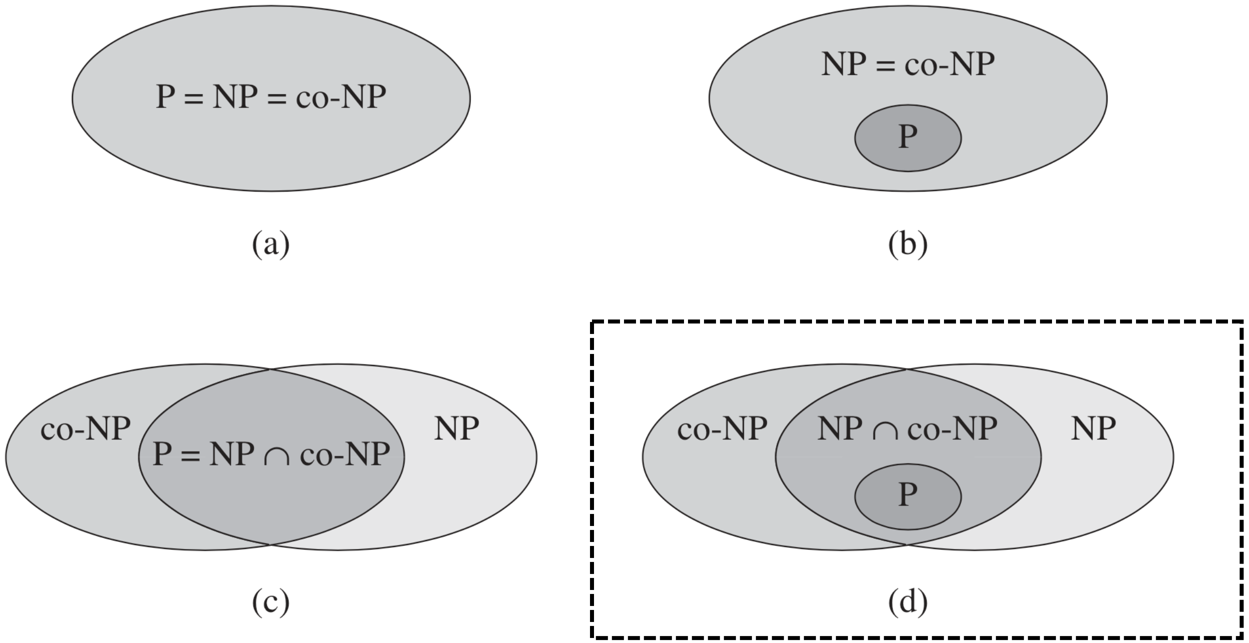
- E.g., Sorting problems.
- E.g., Optimization problems: rod-cutting, matrix-chain multiplication etc.
- E.g., graph problems: minimal spanning tree, single-source shortest paths etc.

Relationship of the three classes

- Any problem in P is also in NP.
  - If we can solve it in polynomial time, then we also can verify a solution in polynomial time.
- For now, we can **<u>believe</u>** that $P \subseteq NP$
- It is still an *open* question.

- If ***any*** NP-complete problem can be solved in polynomial time, then ***every*** problem in NP has a polynomial algorithm
- Most theoretical computer scientists *believe* that NP-complete problems are intractable
  - No one has ever discovered a polynomial time solution to any of them.
  - Yet, <u>no one has found evidence</u> to completely rule out the possibility that NP-complete problems are solvable in polynomial time.

Guesses about the relationship



Here, co-NP is the class of problems for which there is a polynomial-time algorithm that can verify “no” solution (i.e., counterexample).

(a) P = NP = co-NP. Most researchers regard this possibility as the most ***unlikely***.

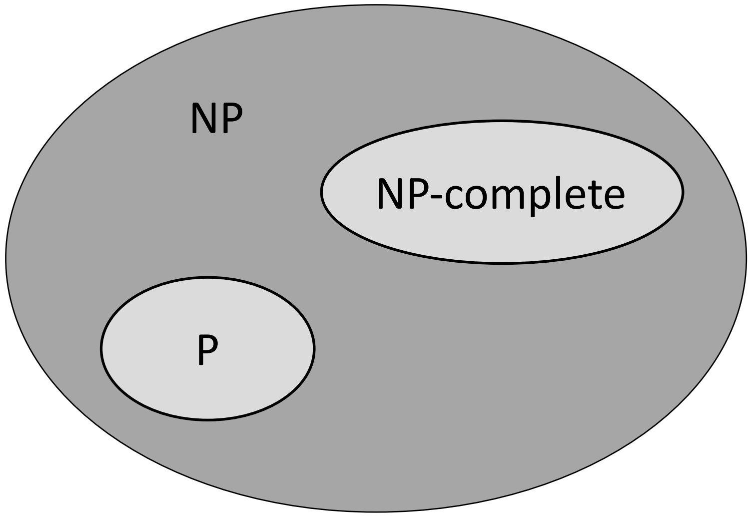
(d) P ≠ NP ∩ co-NP, and NP ≠ co-NP. Most researchers regard this possibility as the most ***likely***.

Guesses about the relationship (cont.)

- A brief summary

Both P and NP-complete are wholly contained in NP

P ∩ NP-complete = ∅



Why such a classification

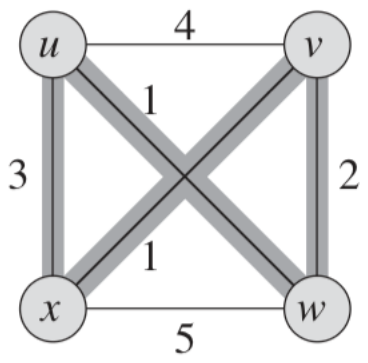
(难解性；不可解性)

- If a problem is NP-complete, it is good evidence for its ***intractability***.

- From an engineering perspective, we prefer developing an *approximation* algorithm or solving a tractable *special case*,
- Rather than searching for a fast algorithm that *solves* the problem exactly.

NP-Complete problems

- **Traveling-salesman problem (TSP)**
- A salesman must visit n cities, represented by a graph of n vertices. (A simple undirected graph in which every pair of distinct vertices is connected by a unique edge)
- The salesman wishes to make a tour (or Hamiltonian cycle), visiting each city exactly once and finishing at the city he starts from.
- The cost to travel from city i to city j is $c(i, j)$, and the salesman wishes to ***minimize*** the total cost of the tour.



Given a sequence of n vertices in the tour, the verification algorithm checks that this sequence contains each vertex exactly once; sums up the edge costs, and checks whether the sum is at most k.

This process can be done in polynomial time. Which shows TSP is in NP.

NP-Complete problems

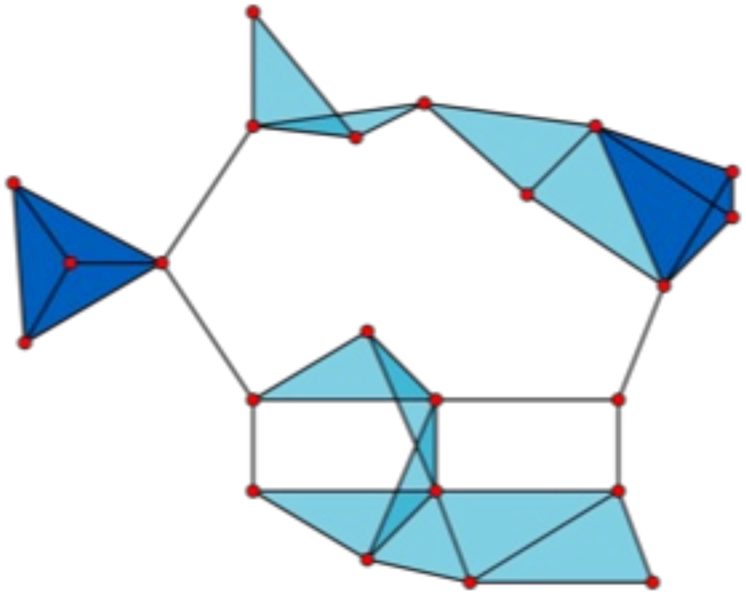
- **The subset-sum problem (SUBSET-SUM)**
- Given a finite set S of positive integers and an integer $t > 0$. Whether there exists a subset $S' \subseteq S$ whose elements sum to t.
- For example:
  if $S = \{1,2,7,14,49,98,343,686,2409,2793,16808,17206,117705,117993\}$
  and $t = 138457$,
  then the subset $S' = \{1,2,7,98,343,686,2409,17206,117705\}$ is a solution.

- To show that SUBSET-SUM is in NP, let S' be the solution, and a verification algorithm can check whether $t = \sum_{s \in S'} s$ in polynomial time.

NP-Complete problems

（团）

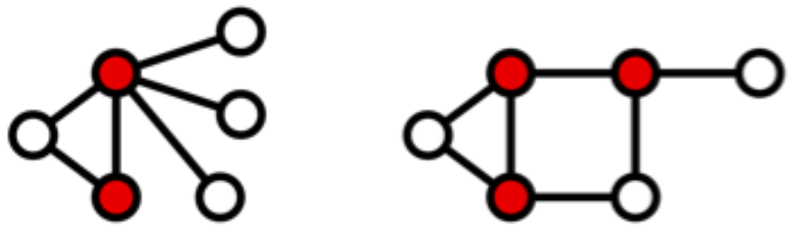
- **The clique problem (CLIQUE)**
- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E.
- Define the size of a clique by the number of vertices it contains.
- The clique problem: find a clique of **maximum size** （最大团）

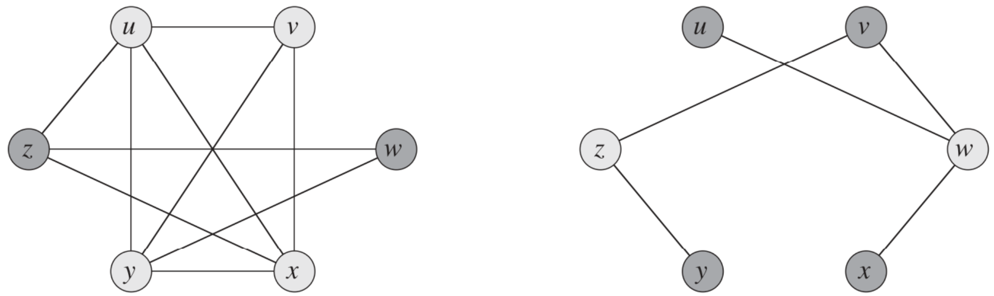


To show CLIQUE is in NP, given a solution V', we can check if V' is a clique in polynomial time by checking whether each edge belongs to E.

NP-Complete problems

- **The vertex-cover problem (VERTEX-COVER)**
- A vertex-cover of an an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both).
- Define the size of a vertex cover by the number of vertices it contains
- The vertex-cover problem: find a vertex cover of **minimum size**.





(a) (b)

Clique of G = $\{u, v, x, y\}$

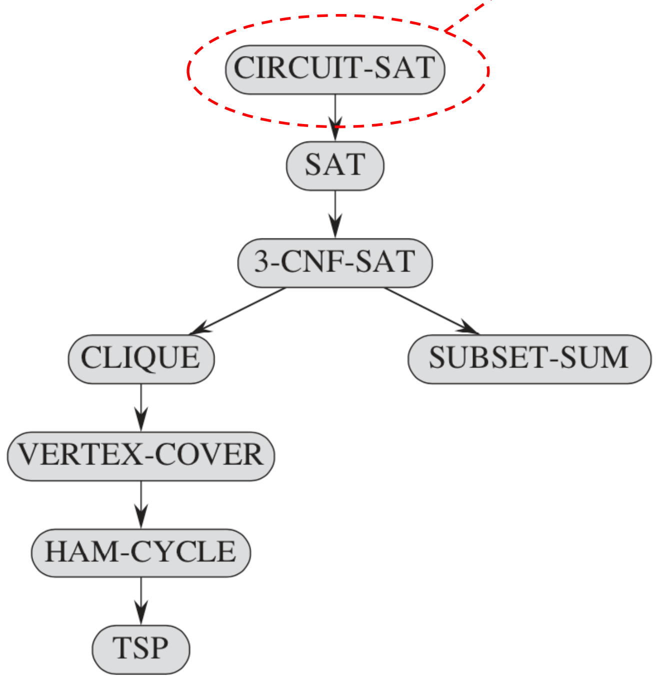
vertex-cover of $\bar{G}$ = $\{w, z\}$

How to prove NP-completeness

- Use the technique ***polynomial-time reduction***.
- It shows one problem is at least as hard as another, within a polynomial-time factor.
- A problem L_1 is ***polynomial-time reducible*** to another problem L_2, written $\boldsymbol{L_1 \leq_P L_2}$.
- It means L_1 is not more than a polynomial factor harder than L_2.

- Now, we can first prove the NP-completeness of L_2. If we can prove $L_1 \leq_P L_2$, then the NP-completeness of L_1 naturally follows.

Just need to prove the NP-completeness of this one



TSP is reducible to HAM-CYCLE

All problems in the figure can be reducible to a CIRCUIT-SAT problem

Knowing about NP-complete problems

- Many problems that on the surface seem no harder than sorting, graph searching, or network flow are in fact NP-complete.

- NP-completeness is a statement about how *hard* a problem is, rather than how *easy* it is.

- We are not trying to prove the existence of an efficient algorithm, but instead that no efficient algorithm is likely to exist.