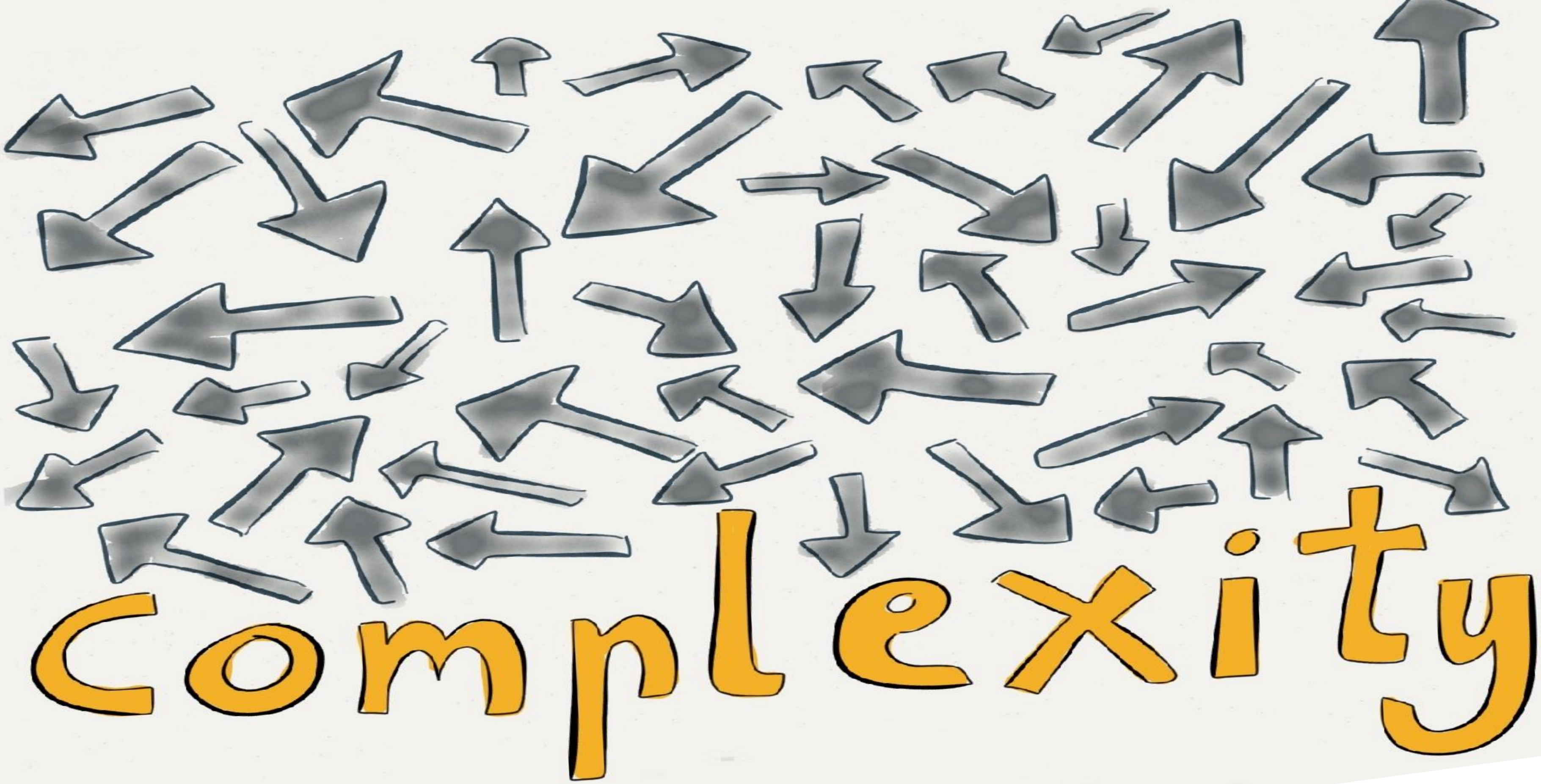


NP Completeness (3)

Dec 20th, 2018

Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學 National Taiwan University

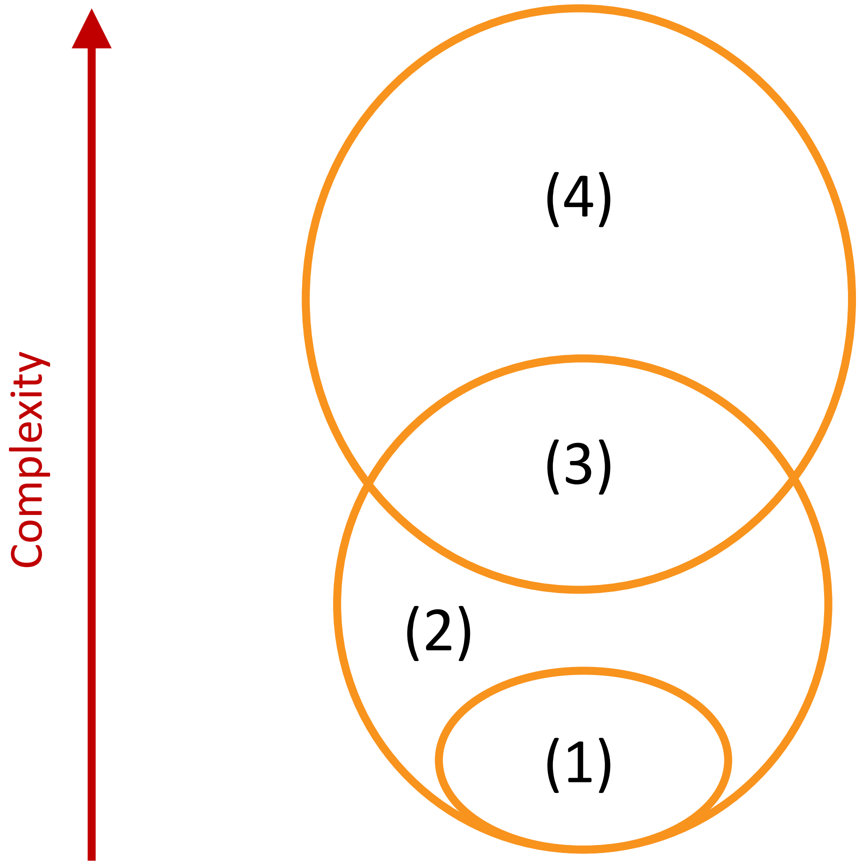
Slides credited from Hsueh-I Lu & Hsu-Chun Hsiao

Outline

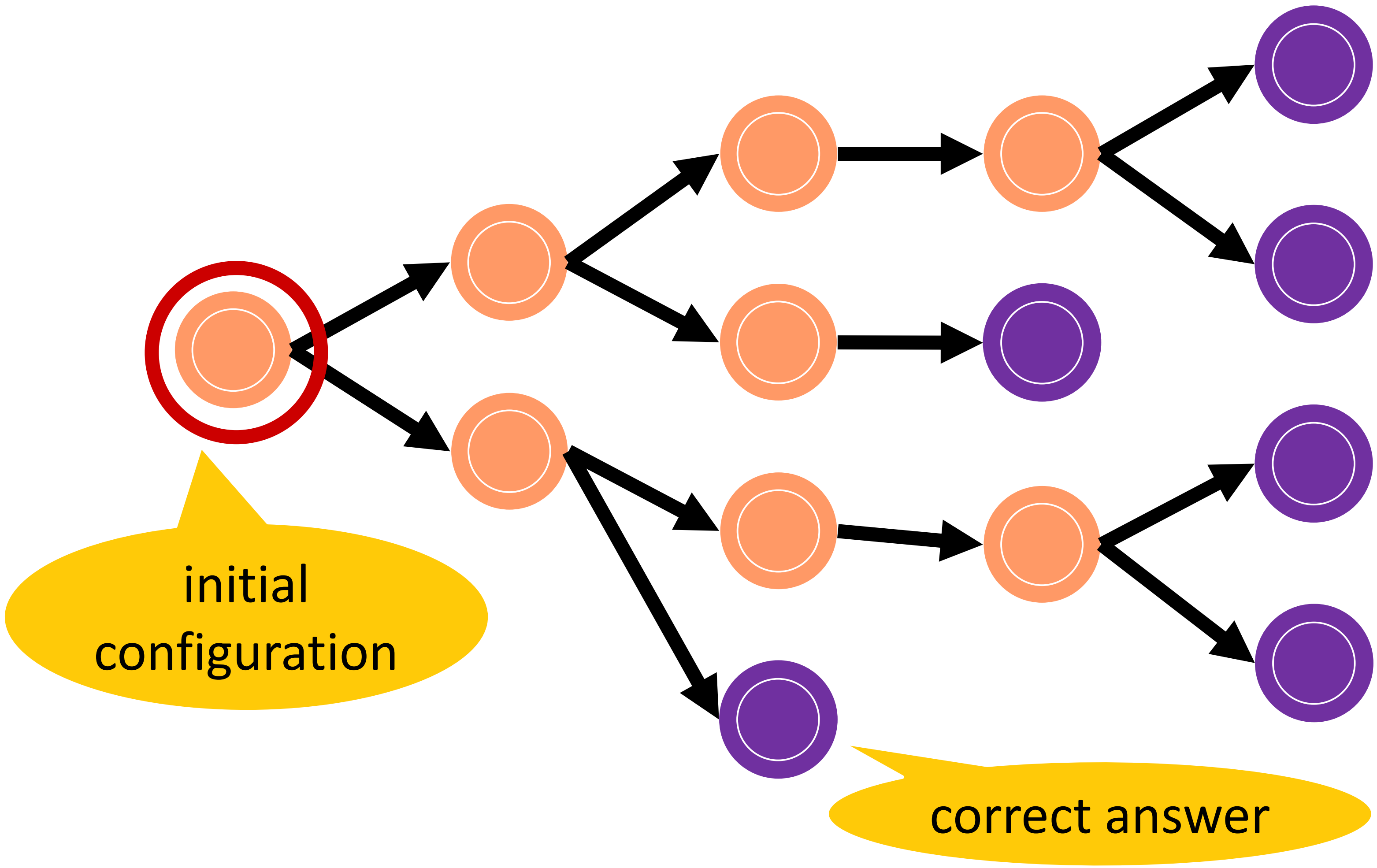
- Polynomial-Time Verification
- Proving NP-Completeness
  - 3-CNF-SAT
  - Clique
  - Vertex Cover
  - Independent Set
  - Traveling Salesman Problem

P, NP, NP-Complete, NP-Hard

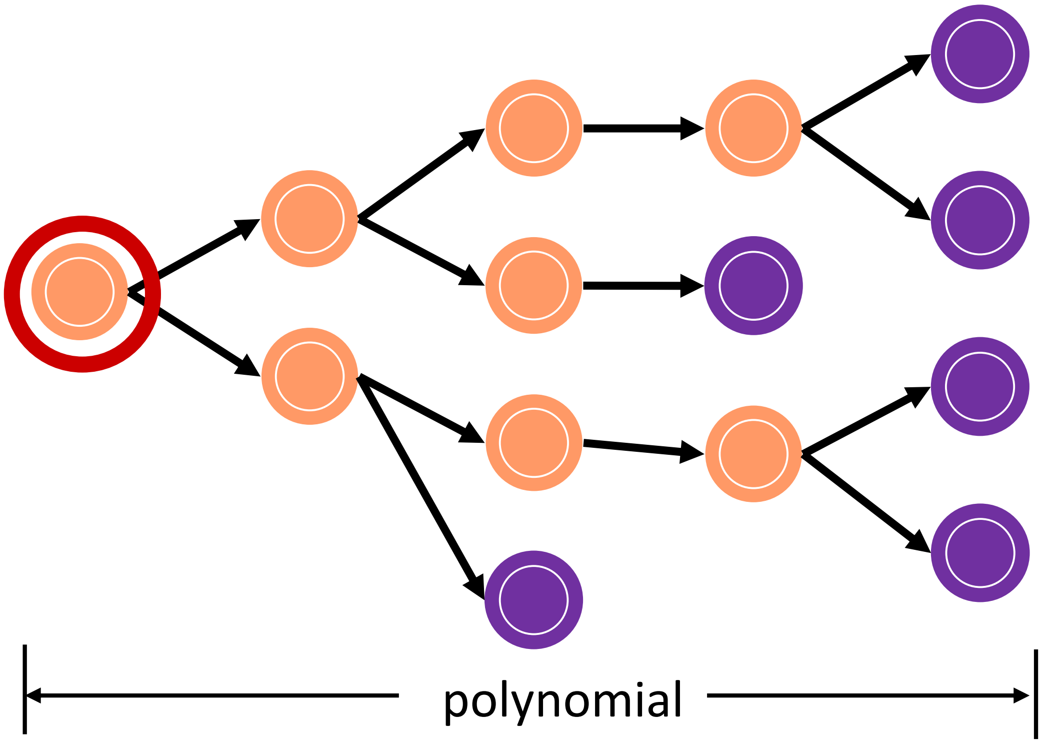
- P ≠ NP



Non-Deterministic Problem Solving



Non-Deterministic Polynomial



"solved" in non-deterministic polynomial time
= "verified" in polynomial time

6

Polynomial-Time Verification

Chapter 34.1 – Polynomial-time

Chapter 34.2 – Polynomial-time verification

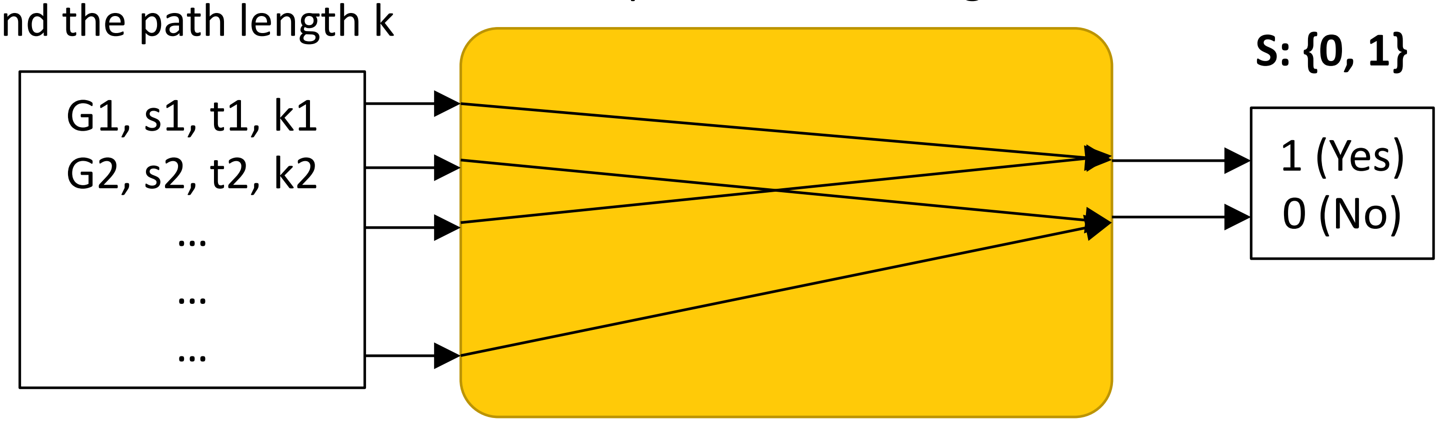
Abstract Problems

- Example of a decision problem, PATH
- **I**: a set of problem instances
- **S**: a set of problem solutions
- **Q**: abstract problem, defined as a *binary relation* on I and S

I: <G, src, dest, k>
All graphs with arbitrary src, dest, and the path length k

Q: PATH
Is there a path with the length ≤ *k*?

S: {0, 1}

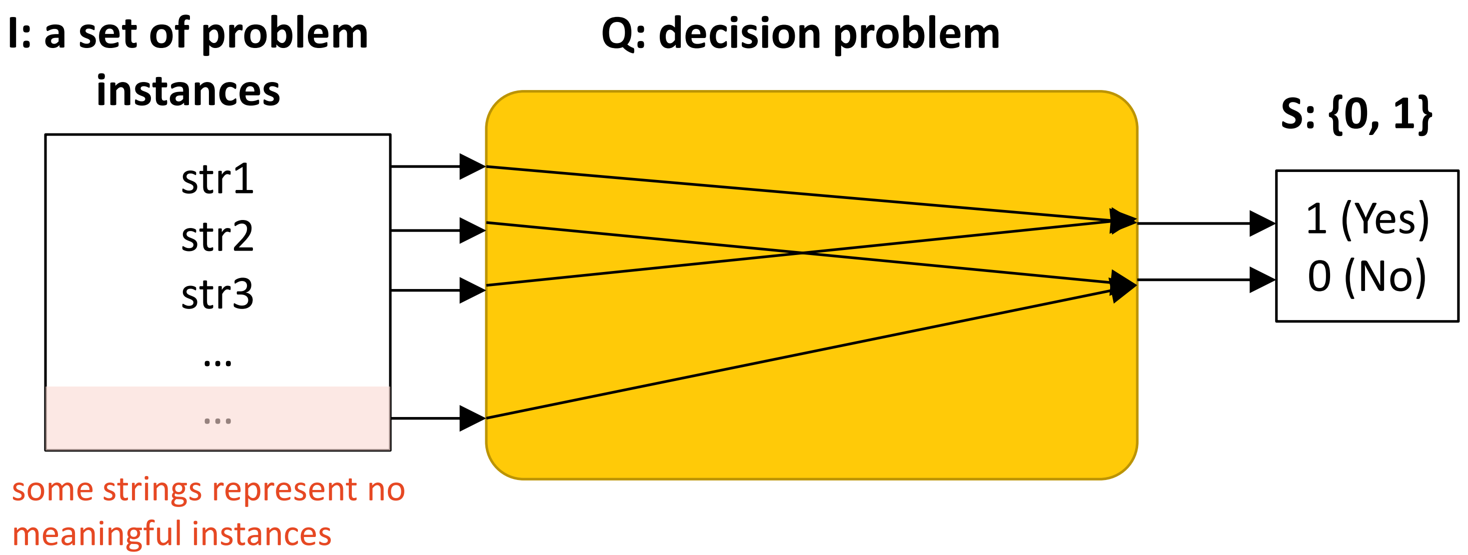


Problem Instance Encoding

- Convert an abstract problem instance into a binary string fed to a computer program

- A concrete problem is **polynomial-time solvable** if there exists an algorithm that solves any concrete instance of length *n* in time $O(n^k)$ for some constant *k*
  - Solvable = can produce a solution

Decision Problem Representation



- I: a set of problem instances $\sum^* = \{\epsilon; 0; 1; 10; 11; 101; 111; \cdots\}$
- Q: a decision problem

  = a language *L* over $\sum = \{0, 1\}$ s.t. $L = \{x \in \{0,1\}^* : Q(x) = 1\}$

以答案為1的instances定義decision problem Q (L = {str1, str3} in this example)

P in Formal Language Framework

A **decision problem** Q can be defined as a **language** *L* over $\sum = \{0, 1\}$ s.t.
$L = \{x \in \{0,1\}^* : Q(x) = 1\}$

- An algorithm A ***accepts*** a string $x \in \{0,1\}^*$ if $A(x) = 1$
- An algorithm A ***rejects*** a string $x \in \{0,1\}^*$ if $A(x) = 0$
- An algorithm A **accepts** a language *L* if A accepts every string $x \in L$
  - If the string is in *L*, A outputs yes.
  - If the string is not in *L*, A may output no or loop forever.
- An algorithm A **decides** a language *L* if A accepts *L* and A rejects every string $x \notin L$
  - For every string, A can output the correct answer.

P in Formal Language Framework

- **Class P:** a class of decision problems *solvable* in polynomial time
- Given an instance x of a decision problem Q, its solution Q(x) (i.e., YES or NO) can be found in polynomial time
- An alternative definition of P:

$$\boxed{P = \{L \subseteq \{0,1\}^* \mid \text{there exists an algorithm that } \textbf{decides } \boldsymbol{L} \textbf{ in polynomial time}\}}$$

- P is the class of language that can be accepted in polynomial time

$$\boxed{P = \{L \mid L \text{ is accepted by a polynomial algorithm}\}}$$

Hamiltonian-Cycle Problem

- Problem: find a cycle that visits each vertex exactly once
- Formal language:

  HAM-CYCLE = {<G> | G has a Hamiltonian cycle}

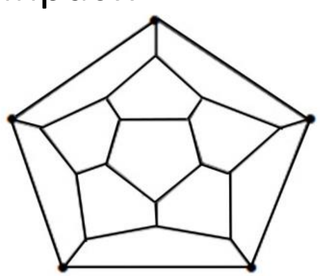
  - Is this language decidable? Yes
  - Is this language decidable in polynomial time? Probably not
- Given a **certificate** – the vertices in order that form a Hamiltonian cycle in G, how much time does it take to **verify** that G indeed contains a Hamiltonian cycle?

Verification Algorithm

- **Verification algorithms** verify memberships in language

HAM-CYCLE = {<G> | G has a Hamiltonian cycle}

Input *x*



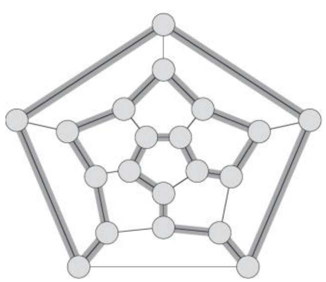
Verification Algorithm

Is *y* a Hamiltonian cycle in the graph (encoded in *x*)?

YES

x is in HAM-CYCLE

Certificate *y*



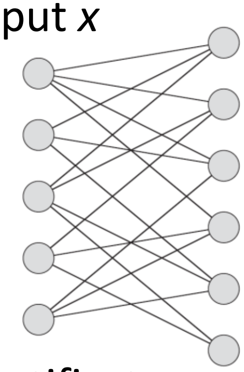
There exists a certificate for each YES instance

Verification Algorithm

- **Verification algorithms** verify memberships in language

HAM-CYCLE = {<G> | G has a Hamiltonian cycle}

Input *x*



Certificate *y*

??

Verification Algorithm

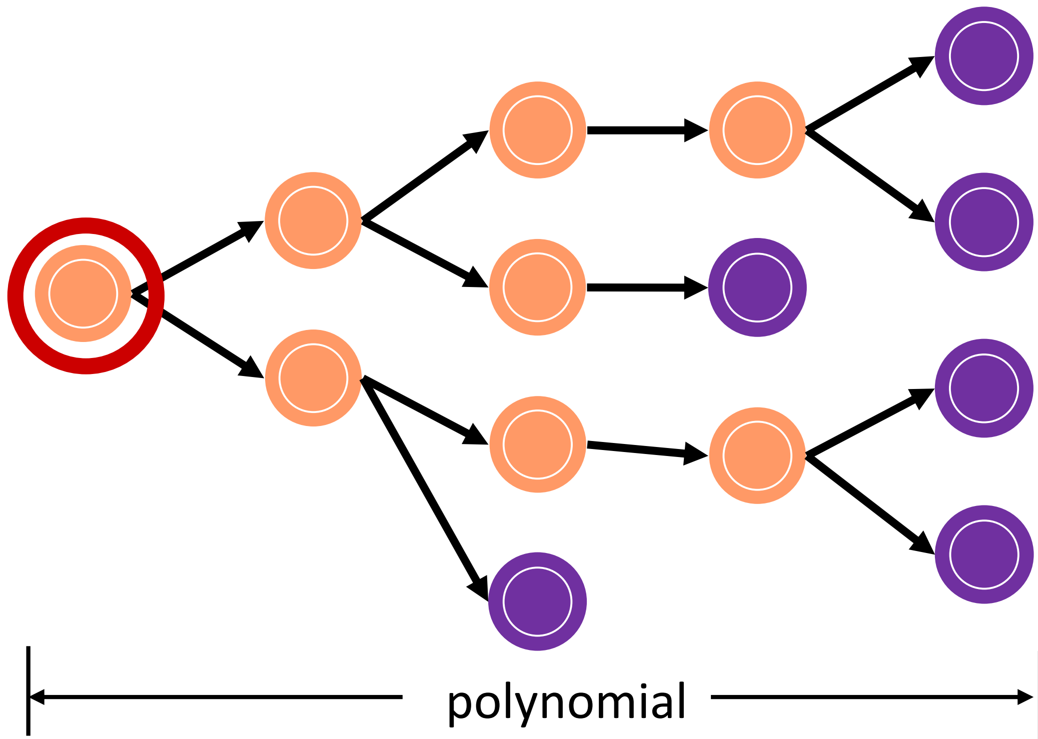
Is *y* a Hamiltonian cycle in the graph (encoded in *x*)?

NO

No conclusion

There exists no certificate for NO instance

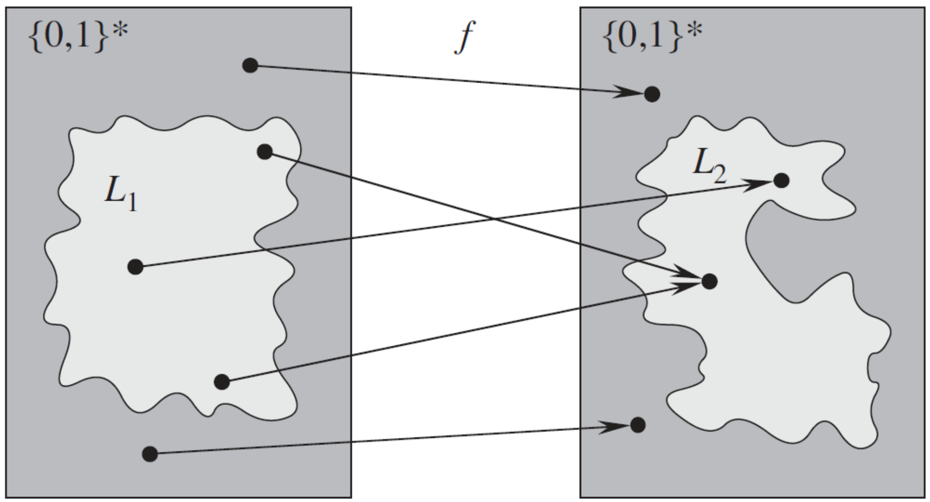
Non-Deterministic Polynomial

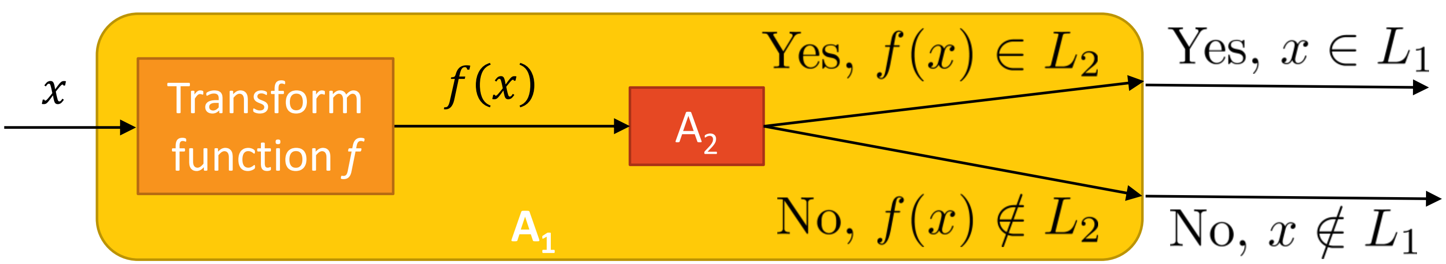


“solved” in non-deterministic polynomial time
= “verified” in polynomial time

Polynomial-Time Reducible

- If $L_1, L_2 \subset \{0,1\}^*$ are languages s.t. $L_1 \leq_p L_2$, then $L_2 \in$ P implies $L_1 \in$ P.

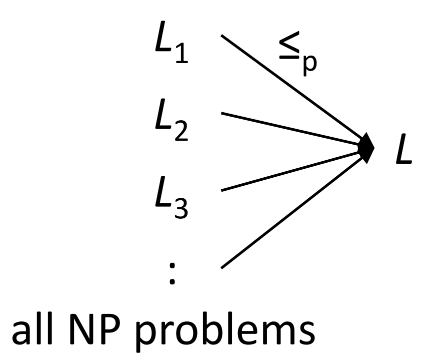


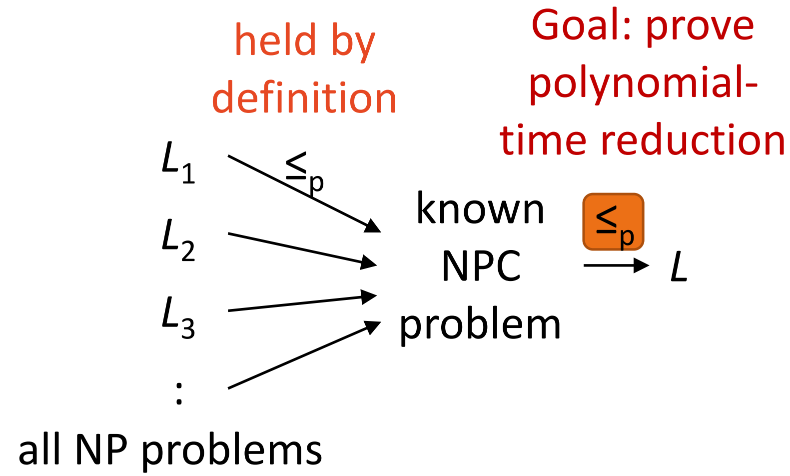


Proving NP-Completeness

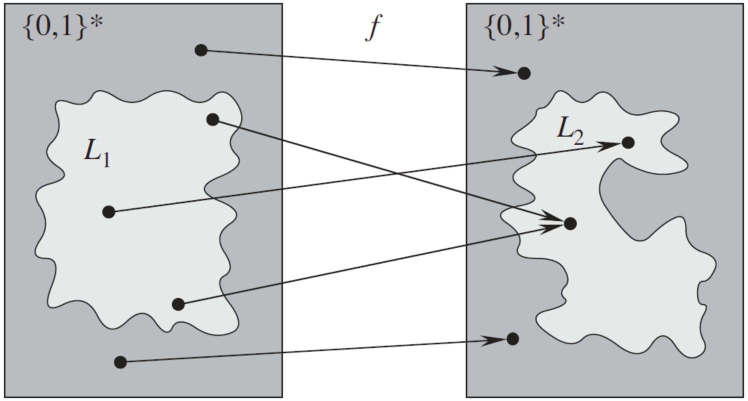
- NP-Complete (NPC): class of decision problems in both NP and NP-hard
- In other words, a decision problem L is NP-complete if
  1. $L \in$ NP
  2. $L \in$ NP-hard (that is, $L' \leq_p L$ for every $L' \in$ NP)

How to prove L is NP-hard ?





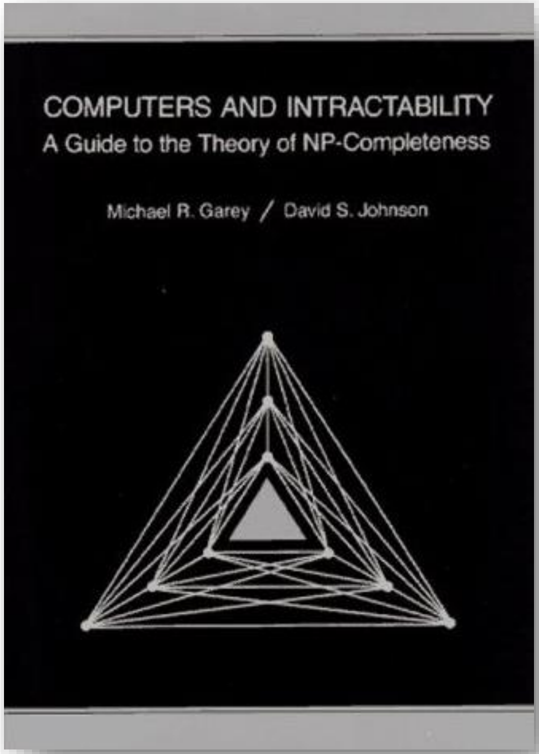
Proving NP-Completeness



- $L \in$ NPC iff $L \in$ NP and $L \in$ NP-hard
- Proof of L in NPC:
  - Prove L ∈ NP
  - Prove L ∈ NP-hard
    1) Select a known NPC problem C
    2) Construct a reduction f transforming every instance of C to an instance of L
    3) Prove that $x \in C \iff f(x) \in L, \forall x \in \{0,1\}^*$
    4) Prove that f is a polynomial time transformation

More NP-Complete Problems

- "Computers and Intractability" by Garey and Johnson includes more than 300 NP-complete problems
  - All except SAT are proved by Karp's polynomial-time reduction



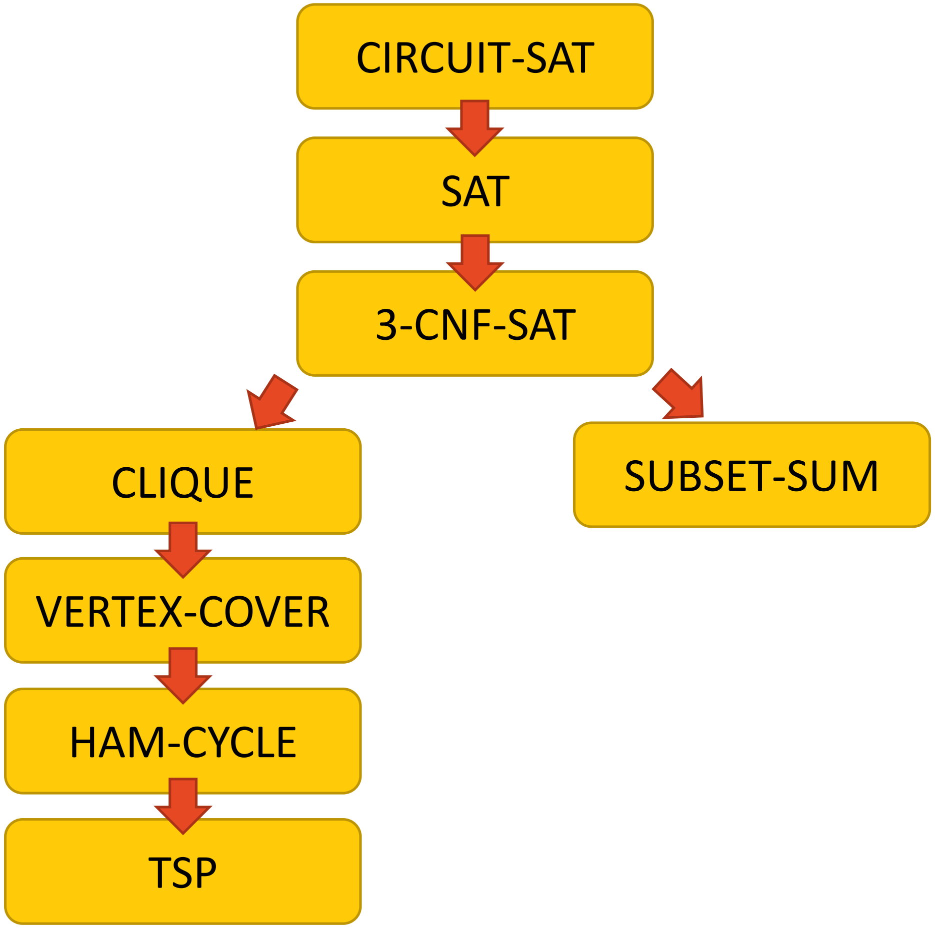
20

Proving NP-Completeness

Chapter 34.5 – NP-complete problems

Roadmap for NP-Completeness

- A → B: A $\leq_p$ B



3-CNF-SAT Problem

- 3-CNF-SAT: Satisfiability of Boolean formulas in 3-*conjunctive normal form* (3-CNF)

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

  - 3-CNF = AND of clauses, each of which is the OR of exactly 3 distinct literals
  - A literal is an occurrence of a variable or its negation, e.g., x_1 or $\neg x_1$

$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$ → satisfiable

3-CNF-SAT

3-CNF-SAT = {Φ | Φ is a Boolean formula in 3-conjunctive normal form (3-CNF) with a satisfying assignment }

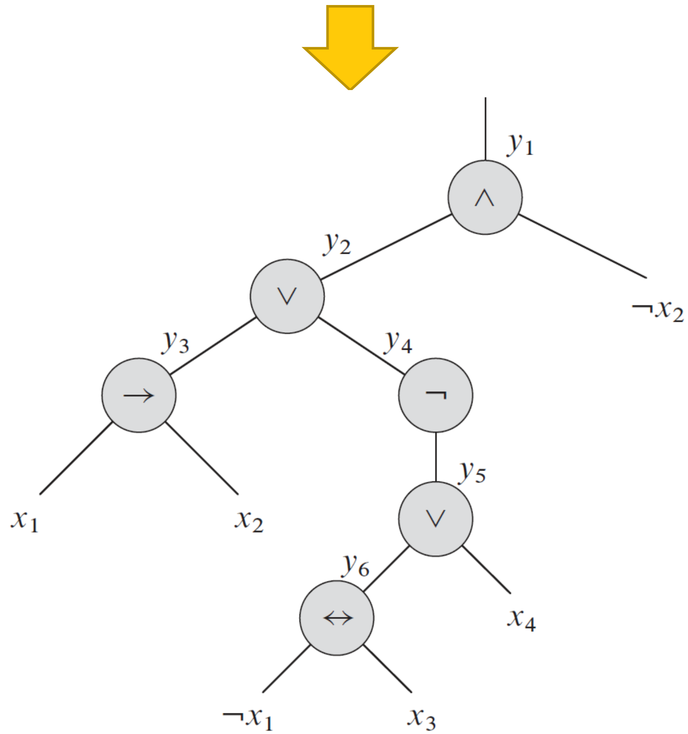
- Is 3-CNF-SAT ∈ NP-Complete?
- To prove that 3-CNF-SAT is NP-Complete, we show that
  - 3-CNF-SAT ∈ NP
  - 3-CNF-SAT ∈ NP-hard (SAT $\leq_p$ 3-CNF-SAT)
    1) SAT is a known NPC problem
    2) Construct a reduction *f* transforming every SAT instance to an 3-CNF-SAT instance
    3) Prove that x ∈ SAT iff *f*(x) ∈ 3-CNF-SAT
    4) Prove that *f* is a polynomial time transformation

We focus on the reduction construction from now on, but remember that a full proof requires showing that all other conditions are true as well

SAT $\leq_p$ 3-CNF-SAT

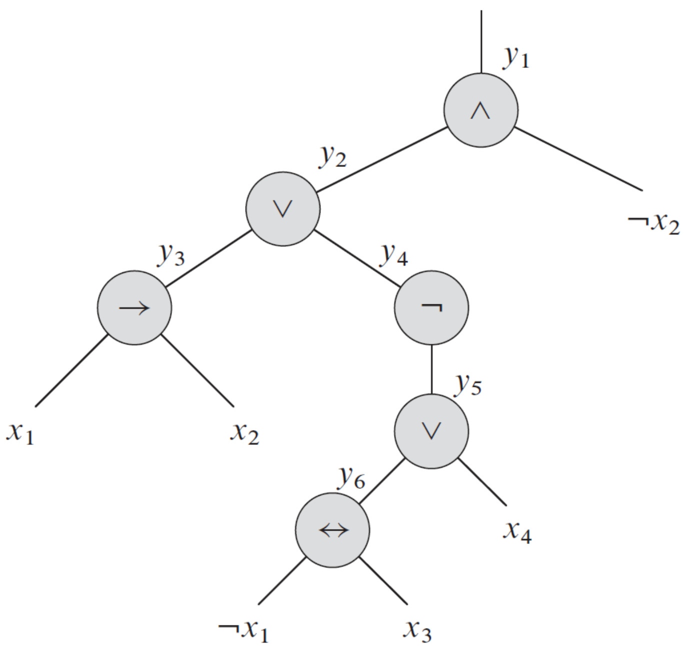
a) Construct a binary parser tree for an input formula Φ and introduce a variable y_i for the output of each internal node

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$



SAT $\leq_p$ 3-CNF-SAT

b) Rewrite Φ as the AND of the root variable and clauses describing the operation of each node



$$
\begin{aligned}
\phi' = y_1 & \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
& \wedge (y_4 \leftrightarrow \neg y_5) \\
& \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\end{aligned}
$$

SAT $\leq_p$ 3-CNF-SAT

$$\phi' = y_1 \wedge \boxed{(y_1 \leftrightarrow (y_2 \wedge \neg x_2))}$$
$$\wedge (y_2 \leftrightarrow (y_3 \vee y_4))$$
$$\wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2))$$
$$\wedge (y_4 \leftrightarrow \neg y_5)$$
$$\wedge (y_5 \leftrightarrow (y_6 \vee x_4))$$
$$\wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))$$

c) Convert each clause Φ_i' to CNF

- Construct a truth table for each clause Φi'
- Construct the disjunctive normal form for ¬Φi'
- Apply DeMorgan's Law to get the CNF formula Φi''

y_1	y_2	x_2	Φ_1'	¬Φ_1'
1	1	1	0	**1**
1	1	0	1	**0**
1	0	1	0	**1**
1	0	0	0	**1**
0	1	1	1	**0**
0	1	0	0	**1**
0	0	1	1	**0**
0	0	0	1	**0**

$$\neg\phi'_1 = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

$$\Downarrow$$

$$\phi'_1 = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

$$\boxed{\begin{array}{l} \neg(a \wedge b) = \neg a \vee \neg b \\ \neg(a \vee b) = \neg a \wedge \neg b \end{array}}$$

SAT $\leq_p$ 3-CNF-SAT

d) Construct Φ''' in which each clause C_i exactly 3 distinct literals
   - 3 distinct literals: $C_i = l_1 \vee l_2 \vee l_3$
   - 2 distinct literals: $C_i = l_1 \vee l_2$

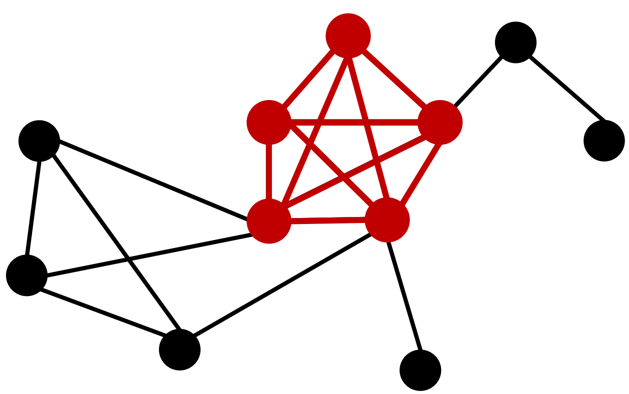
     $C_i = l_1 \vee l_2 = (l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$
   - 1 literal only: $C_i = l$

     $C_i = l = (l \vee p \vee q) \wedge (l \vee \neg p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee \neg q)$

- Φ''' is satisfiable iff Φ is satisfiable
- All transformation can be done in polynomial time
- → 3-CNF-SAT is NP-Complete

Clique Problem

- A clique in G = (V, E) is a *complete* subgraph of G
  - Each pair of vertices in a clique is connected by an edge in *E*
  - Size of a clique = # of vertices it contains
- Optimization problem: find a max clique in G
- Decision problem: is there a clique with size larger than *k*



Does G contain a clique of size 4?	Yes
Does G contain a clique of size 5?	Yes
Does G contain a clique of size 6?	No

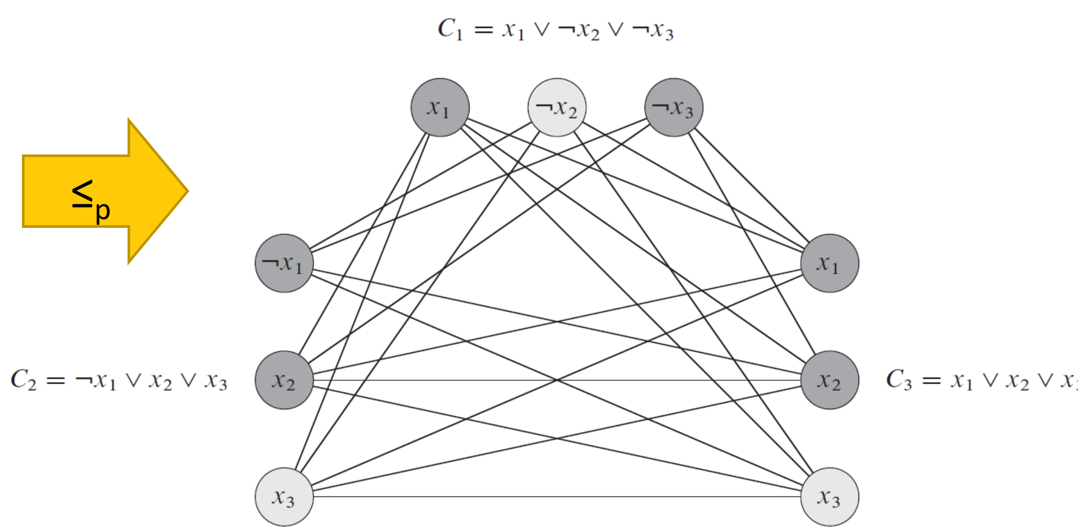
CLIQUE ∈ NP-Complete

CLIQUE = {<G, *k*>: G is a graph containing a clique of size *k*}

- Is CLIQUE ∈ NP-Complete? 3-CNF-SAT $\leq_p$ CLIQUE
- Construct a reduction *f* transforming every 3-CNF-SAT instance to a CLIQUE instance
- **a graph G s.t. Φ with *k* clauses is satisfiable ⇔ G has a clique of size k**

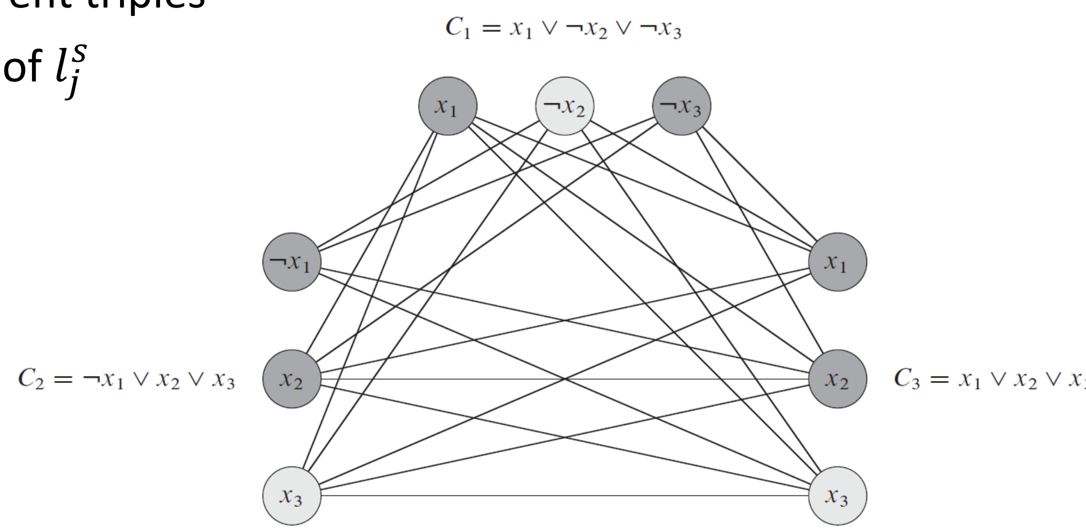
$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

$\leq_p$



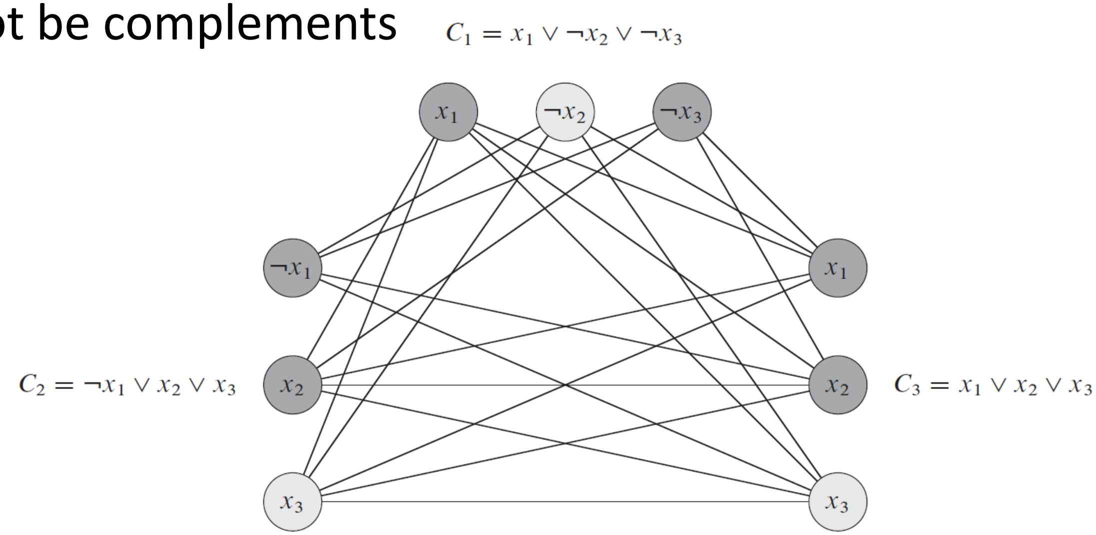
CLIQUE ∈ NP-Complete

- <u>Polynomial-time reduction</u>:
- Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a Boolean formula in 3-CNF with *k* clauses, and each C_r has exactly 3 distinct literals l_1^r, l_2^r, l_3^r
- For each $C_r = (l_1^r \vee l_3^r \vee l_3^r)$, introduce a triple of vertices v_1^r, v_2^r, v_3^r in V
- Build an edge between v_i^r, v_j^s if both of the following hold:
  - v_i^r and v_j^s are in different triples
  - l_i^r is not the negation of l_j^s



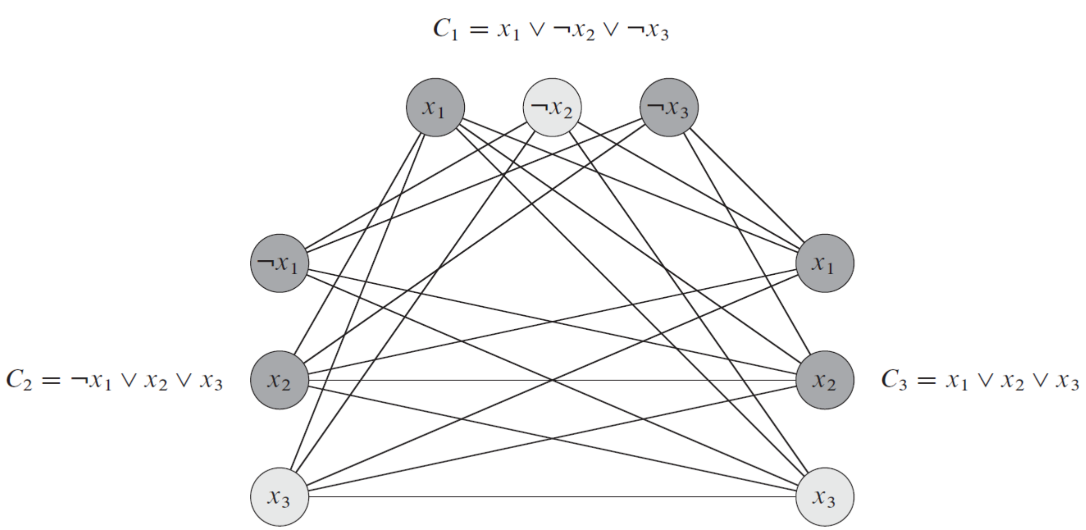
3-CNF-SAT $\leq_p$ CLIQUE

- <u>Correctness proof</u>: **Φ is satisfiable → G has a clique of size *k***
- If Φ is satisfiable
- → Each C_r contains at least one $l_i^r = 1$ and such literal corresponds to v_i^r
- → Pick a TRUE literal from each C_r forms a set of V' of *k* vertices
- → For any two vertices $v_i^r, v_j^s \in V'(r \neq s)$, edge $(v_i^r, v_j^s) \in E$, because $l_i^r = l_j^s = 1$ and they cannot be complements



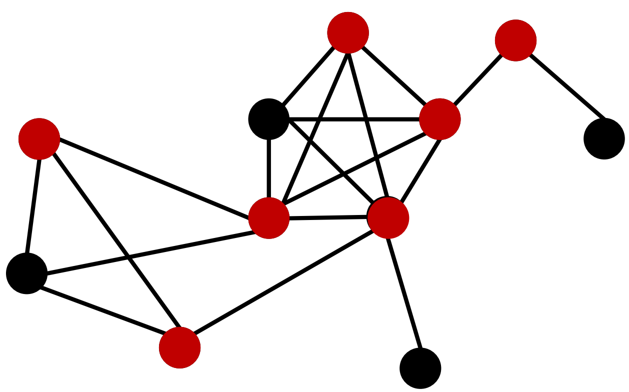
3-CNF-SAT $\leq_p$ CLIQUE

- <u>Correctness proof</u>: **G has a clique of size *k* → Φ is satisfiable**
- G has a clique V' of size k
- → V' contains exactly one vertex per triple since no edges connect vertices in the same triple
- → Assign 1 to each l_i^r where $v_i^r \in V'$ s.t. each C_r is satisfiable, and so is Φ



Vertex Cover Problem

- A vertex cover of G = (V, E) is a subset V' ⊆ V s.t. if (w, v) ∈ E, then w ∈ V' or v ∈ V'
  - A vertex cover "covers" every edge in G
- Optimization problem: find a minimum size vertex cover in G
- Decision problem: is there a vertex cover with size smaller than *k*



Does G have a vertex cover of size 11? Yes

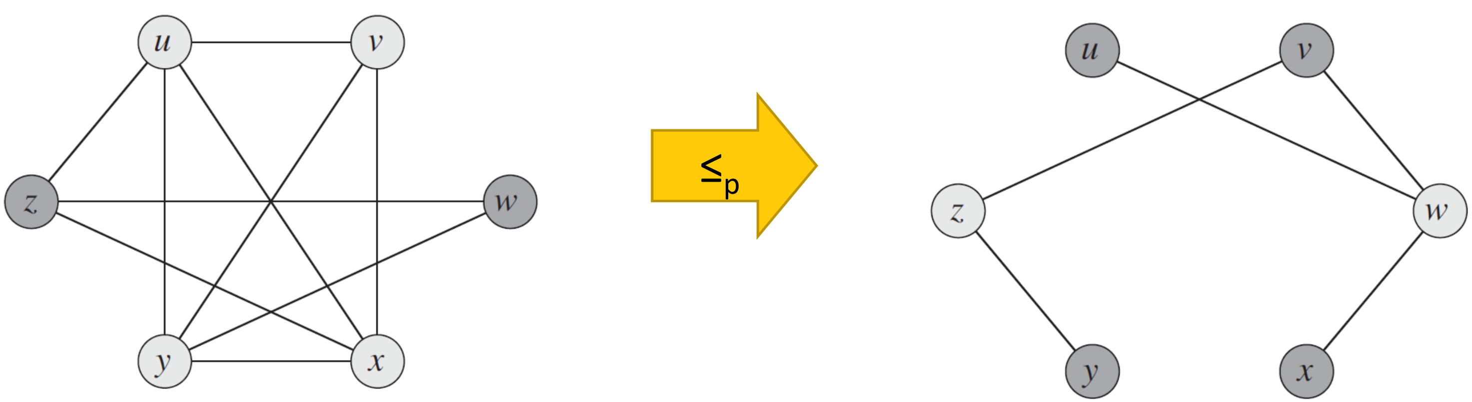
Does G have a vertex cover of size 7? Yes

Does G have a vertex cover of size 6? No

VERTEX-COVER ∈ NP-Complete

VERTEX-COVER = {<G, *k*>: G is a graph containing a vertex cover of size *k*}

- Is VERTEX-COVER ∈ NP-Complete? CLIQUE $\leq_p$ VERTEX-COVER
- Construct a reduction *f* transforming every CLIQUE instance to a VERTEX-COVER instance (polynomial-time reduction)
  - Compute the *complement* of G
  - Given G = <V, E>, G_c is defined as <V, E_c> s.t. $E_c = \{(u,v) \mid (u,v) \notin E\}$
- **a graph G has a clique of size *k* ⇔ G_c has a vertex cover of size |V| - *k***

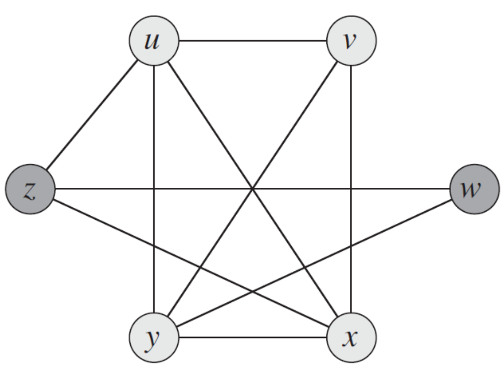


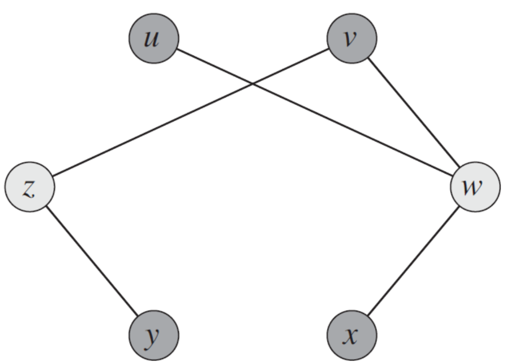
CLIQUE $\leq_p$ VERTEX-COVER

- <u>Correctness proof</u>:

a graph G has a clique of size *k* → G_c has a vertex cover of size |V| - *k*

- If G has a clique V' ⊆ V with |V'| = *k*
- → for all $(w, v) \in E_c$, at least one of w or $v \notin V'$
- → $w \in V - V'$ or $v \in V - V'$ (or both)
- → edge (w, v) is covered by $V - V'$
- → $V - V'$ forms a vertex cover of G_c, and |V - V'| = |V| - *k*



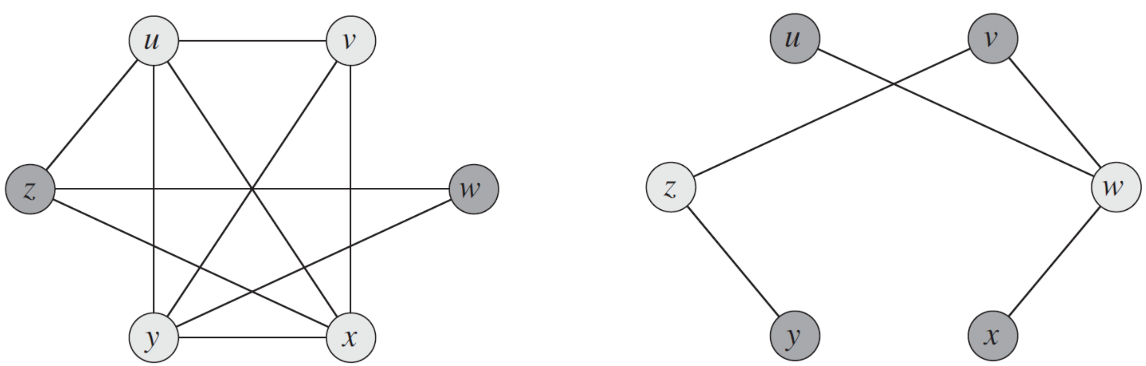


CLIQUE $\leq_p$ VERTEX-COVER

- Correctness proof:

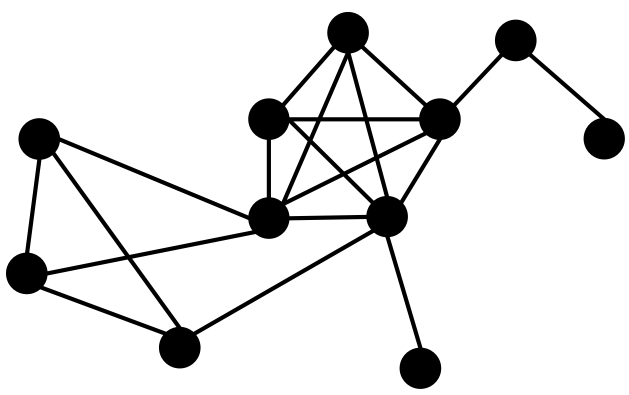
G_c has a vertex cover of size |V| - *k* → a graph G has a clique of size *k*

- If G_c has a vertex cover V' ⊆ V with |V'| = |V| - *k*
- → for all $w, v \in V$, if $(w, v) \in E_c$, then $w \in V'$ or $v \in V'$ or both
- → for all $w, v \in V$, if $w \notin V'$ and $v \notin V'$, $(w, v) \in E$
- → $V - V'$ is a clique where |V - V'| = *k*



Independent-Set Problem

- An independent set of G = (V, E) is a subset V' ⊆ V such that G has no edge between any pair of vertices in V'
  - A vertex cover "covers" every edge in G
- Optimization problem: find a maximum size independent set
- Decision problem: is there an independent set with size larger than *k*



Does G have an independent set of size 1?

Does G have an independent set of size 4?

Does G have an independent set of size 5?

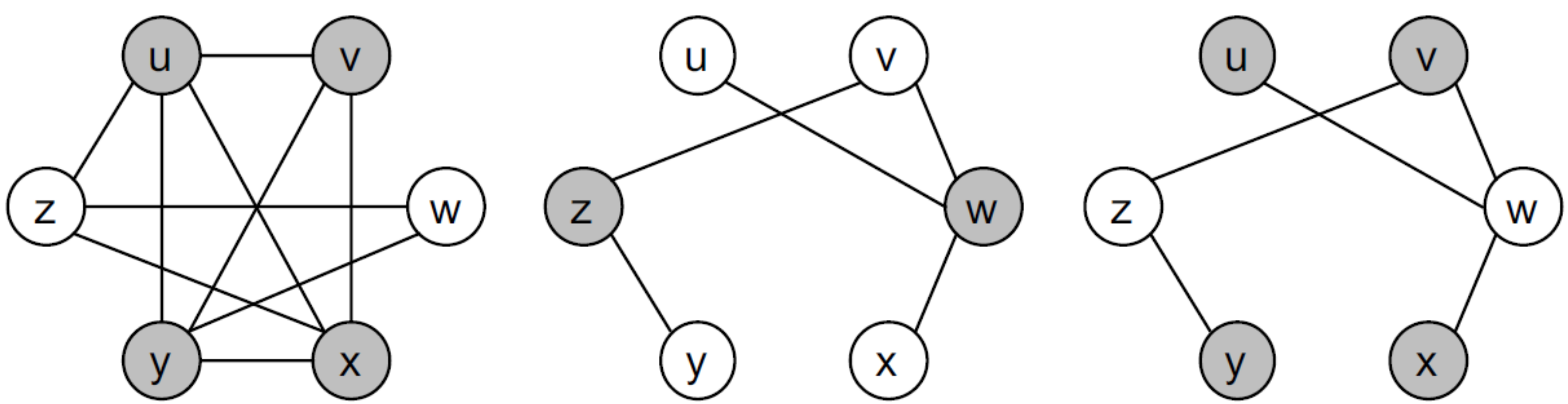
IND-SET ∈ NP-Complete

IND-SET = {<G, *k*>: G is a graph containing an independent set of size *k*}

- Is IND-SET ∈ NP-Complete?
- Practice by yourself (textbook problem 34-1)

CLIQUE, VERTEX-COVER, IND-SET

- The following are equivalent for G = (V, E) and a subset V’ of V:
  1) V' is a clique of G
  2) V-V' is a vertex cover of G_c
  3) V' is an independent set of G_c



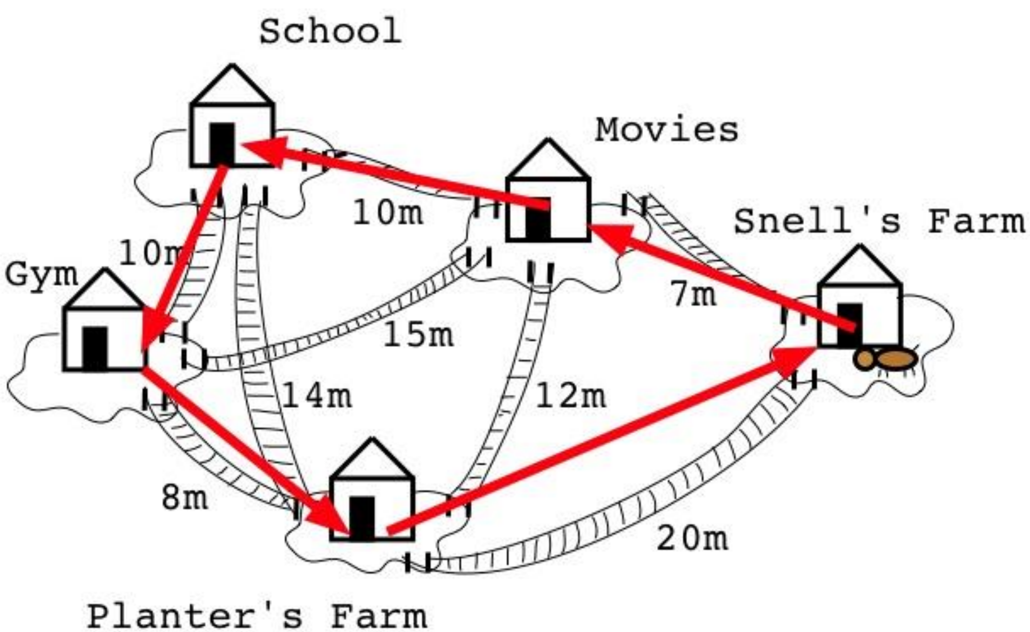
Clique
V’ = {u, v, x, y} in G

Vertex cover
V - V’ = {z, w} in G_c

Independent set
V’ = {u, v, x, y} in G_c

Traveling Salesman Problem (TSP)

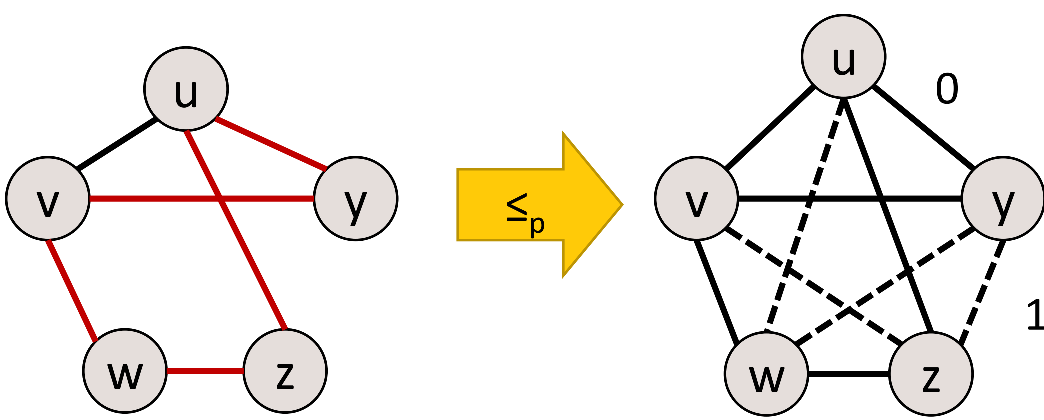
- Optimization problem: Given a set of cities and their pairwise distances, find a tour of lowest cost that visits each city exactly once.
- Decision problem: is there a traveling salesman tour with cost at most *k*



TSP ∈ NP-Complete

TSP = {<G, c, k>: G = (V,E) is a complete graph, c is a cost function for edges, G has a traveling-salesman tour with cost at most k}

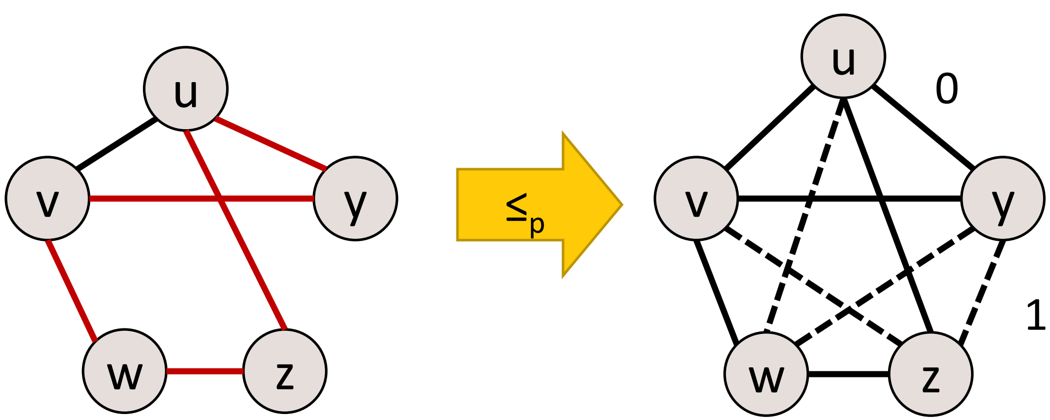
- Is TSP ∈ NP-Complete? HAM-CYCLE $\leq_p$ TSP
- Construct a reduction *f* transforming every HAM-CYCLE instance to a TSP instance (polynomial-time reduction)
- **G contains a Hamiltonian cycle *h* = $<v_1, v_2, \ldots, v_n, v_1>$ ⇔ $<v_1, v_2, \ldots, v_n, v_1>$ is a traveling-salesman tour with cost 0**



$$c(i,j) = \begin{cases} 0 & \text{if } (i,j) \in E \\ 1 & \text{if } (i,j) \notin E \end{cases}$$

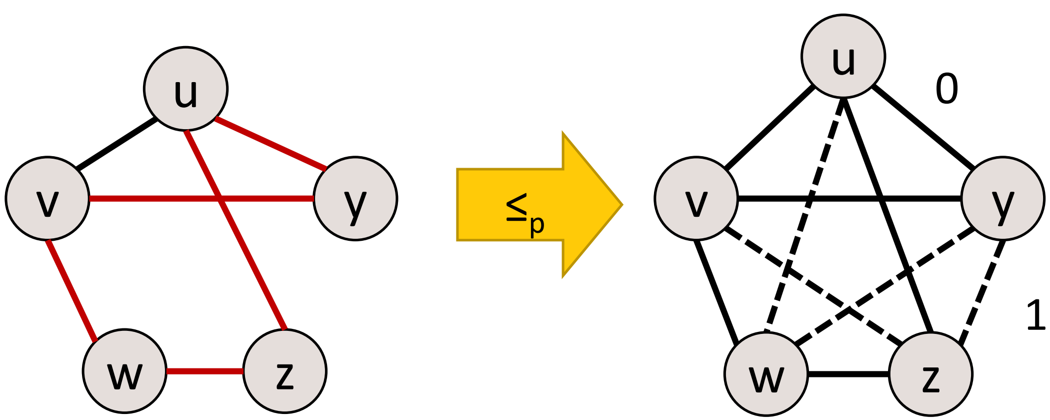
HAM-CYCLE $\leq_p$ TSP

- <u>Correctness proof</u>: ***x* ∈ HAM-CYCLE → *f*(*x*) ∈ TSP**
- If Hamiltonian cycle is $h = \langle v_1, v_2, \ldots, v_n, v_1 \rangle$
- → *h* is also a tour in the transformed TSP instance
- → The distance of the tour *h* is *0* since there are *n* consecutive edges in *E*, and so has distance 0 in *f*(*x*)
- → *f*(*x*) ∈ TSP (*f*(*x*) has a TSP tour with cost ≤ 0)



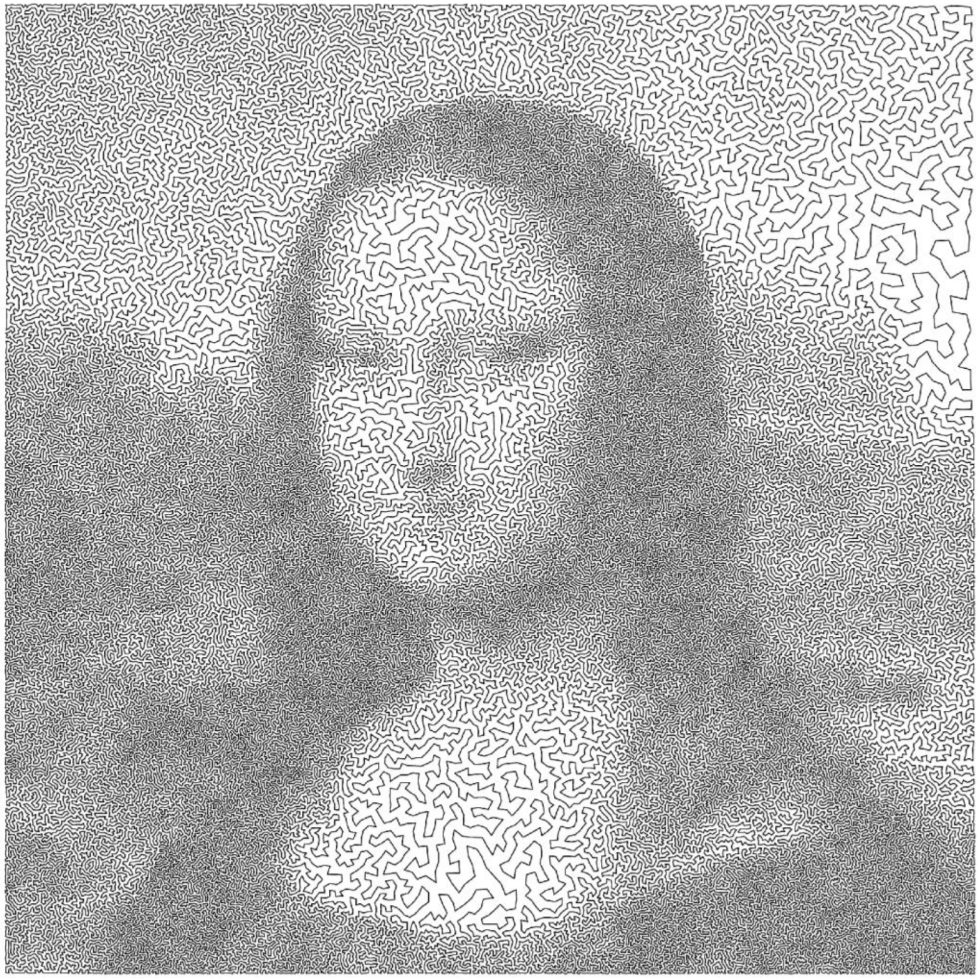
HAM-CYCLE $\leq_p$ TSP

- <u>Correctness proof</u>: ***f*(*x*) ∈ TSP → *x* ∈ HAM-CYCLE**
- **After reduction**, if a TSP tour with cost ≤ 0 as $<v_1, v_2, \ldots, v_n, v_1>$
- → The tour contains only edges in E
- → Thus, $<v_1, v_2, \ldots, v_n, v_1>$ is a Hamiltonian cycle



TSP Challenges

- Mona Lisa TSP: $1,000 Prize for 100,000-city



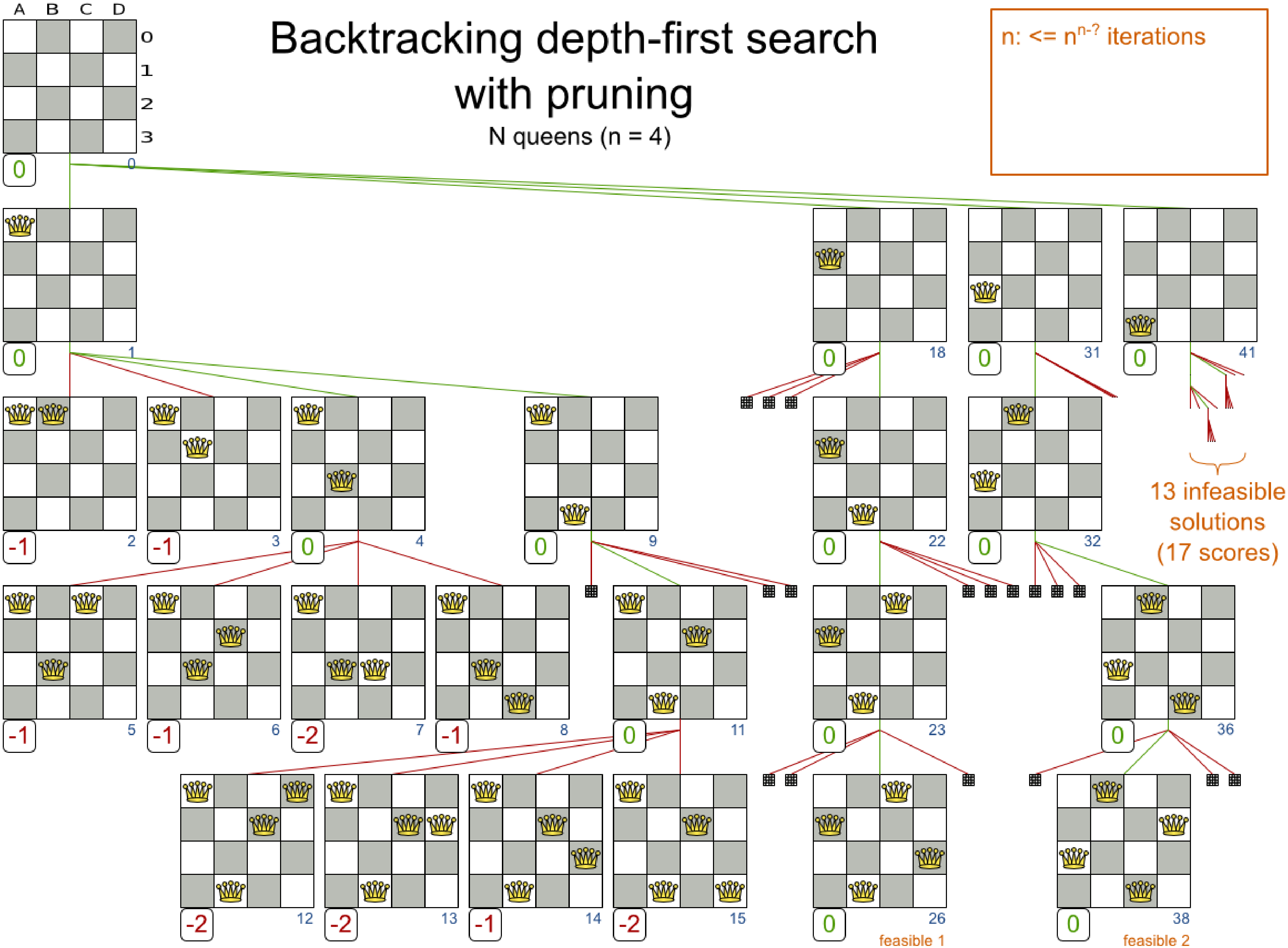
Strategies for NP-Complete/NP-Hard Problems

- NP-complete/NP-hard problems are unlikely to have polynomial-time solutions (unless P = NP), we must sacrifice either **optimality**, **efficiency**, or **generality**
  - **Approximation algorithms:** guarantee to be a fixed percentage away from the optimum
  - **Local search:** simulated annealing (hill climbing), genetic algorithms, etc
  - **Heuristics:** no formal guarantee of performance
  - **Randomized algorithms:** use a randomizer (random number generator) for operation
  - **Pseudo-polynomial time algorithms:** e.g., DP for 0-1 knapsack
  - **Exponential algorithms/Branch and Bound/Exhaustive search:** feasible only when the problem size is small
  - **Restriction:** work on some special cases of the original problem. e.g., the maximum independent set problem in circle graphs

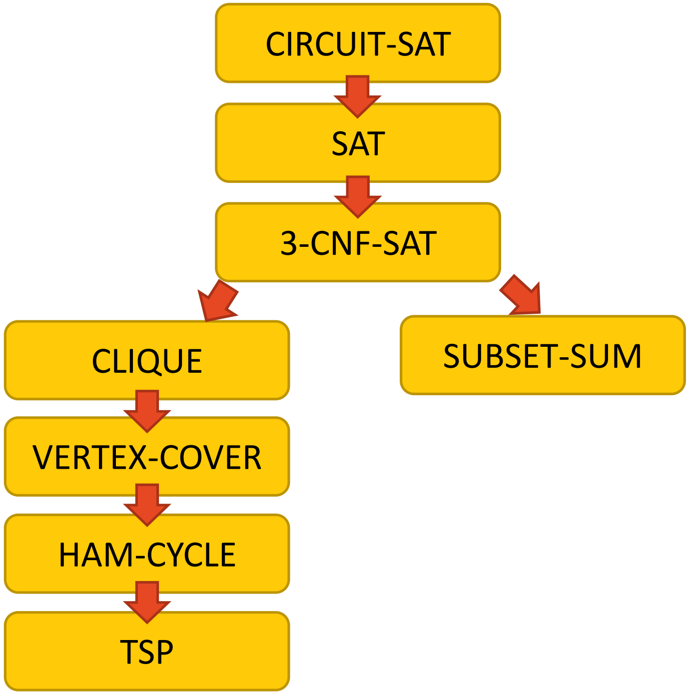
Backtracking depth-first search with pruning

N queens (n = 4)

n: <= $n^{n-?}$ iterations



Concluding Remarks



- Polynomial-time verification
- *L* ∈ NPC iff *L* ∈ NP and *L* ∈ NP-hard
- Step-by-step approach for proving *L* in NPC:
  - Prove *L* ∈ NP
  - Prove *L* ∈ NP-hard
    1) Select a known NPC problem C
    2) Construct a reduction f transforming every instance of C to an instance of L
    3) Prove that $x \in C \iff f(x) \in L, \forall x \in \{0,1\}^*$
    4) Prove that *f* is a polynomial time transformation
- Strategies for NP-complete/NP-hard problems

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw