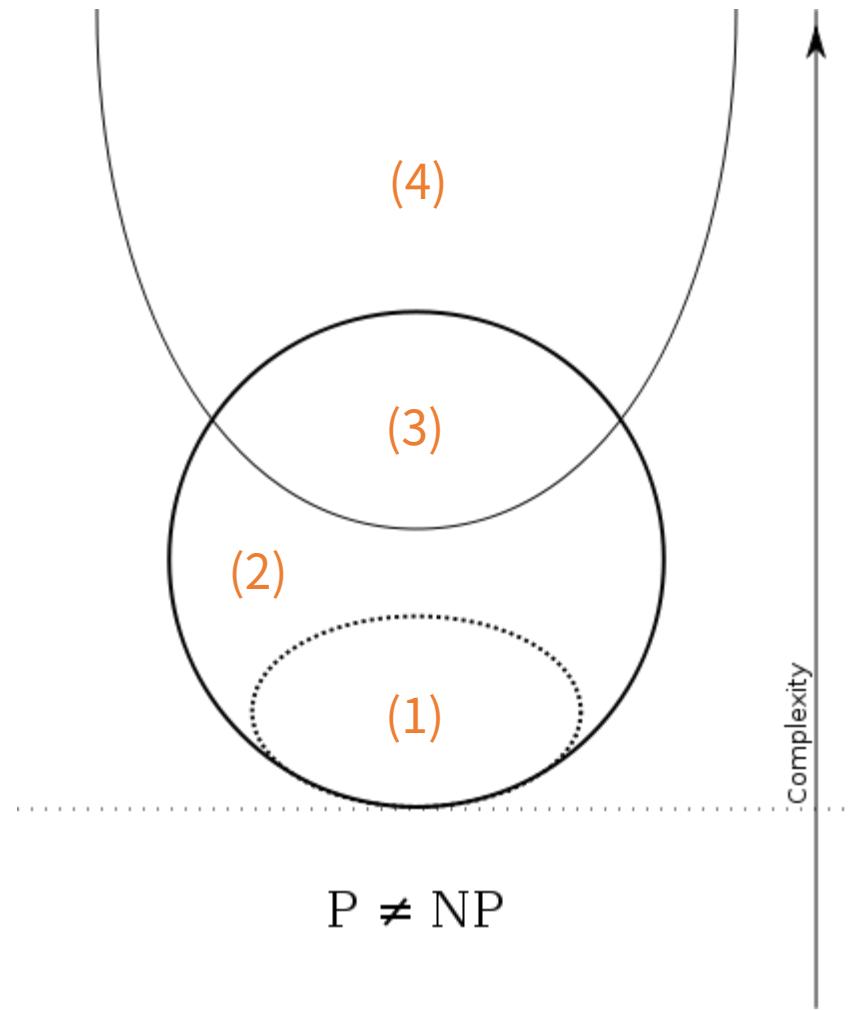


- NP-Completeness Overview
 - Warm up: four color problem
 - Decision vs. optimization
 - Complexity classes
 - Reduction
 - P-time solving vs. verification
- Proving NP-Completeness
 - Formula satisfiability problem
 - 3-CNF-SAT
 - The clique problem
 - The vertex-cover problem
 - The independent-set problem
 - Traveling salesman problem
 - Hamiltonian cycle

3-week Agenda

- Approximation algorithms
 - Vertex Cover
 - TSP
 - 3-CNF-SAT
- Randomized algorithms
 - Karger’s min-cut algorithm
 - Probabilistic data structures

Review: P, NP, NP-Hard, NP-Complete



Complexity classes

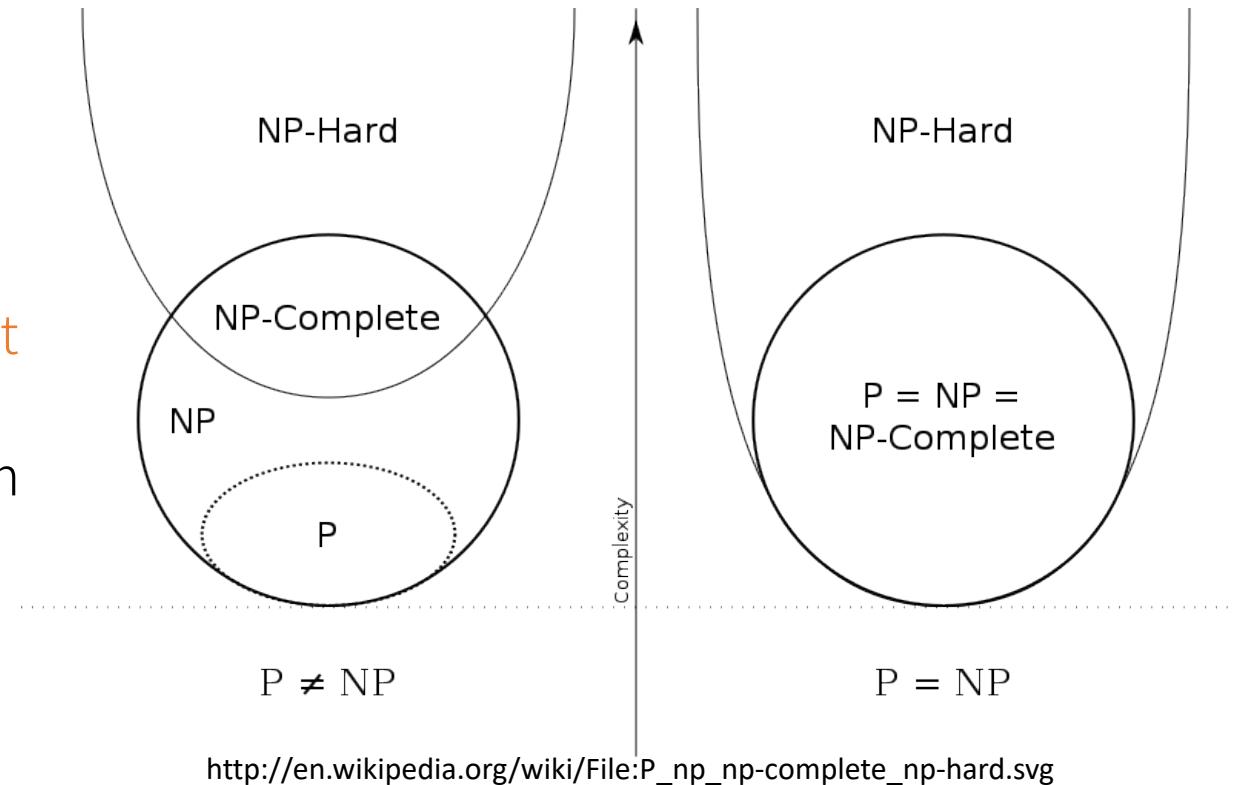
Class P: class of problems that can be solved in $O(n^k)$

Class NP: class of problems that can be verified in $O(n^k)$

Class NP-hard: a class of problems that are "at least as hard as the hardest problems" in NP

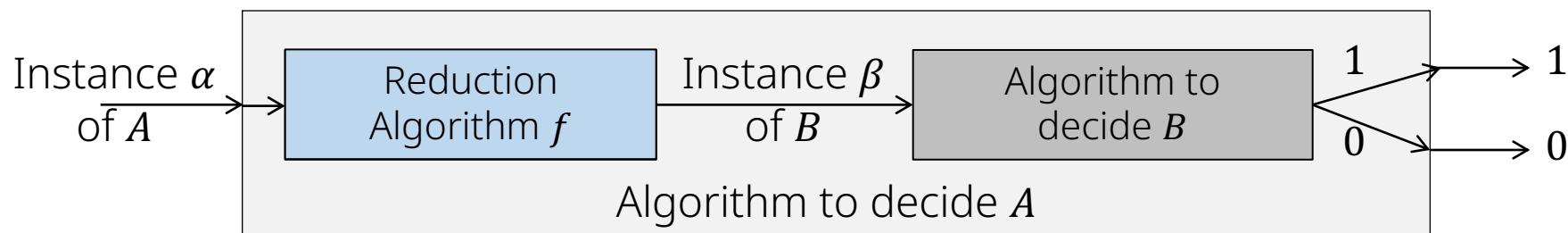
Class NP-complete (NPC): class of problems in both NP and NP-hard

Hardness relationship can be determined via **polynomial-time reduction**



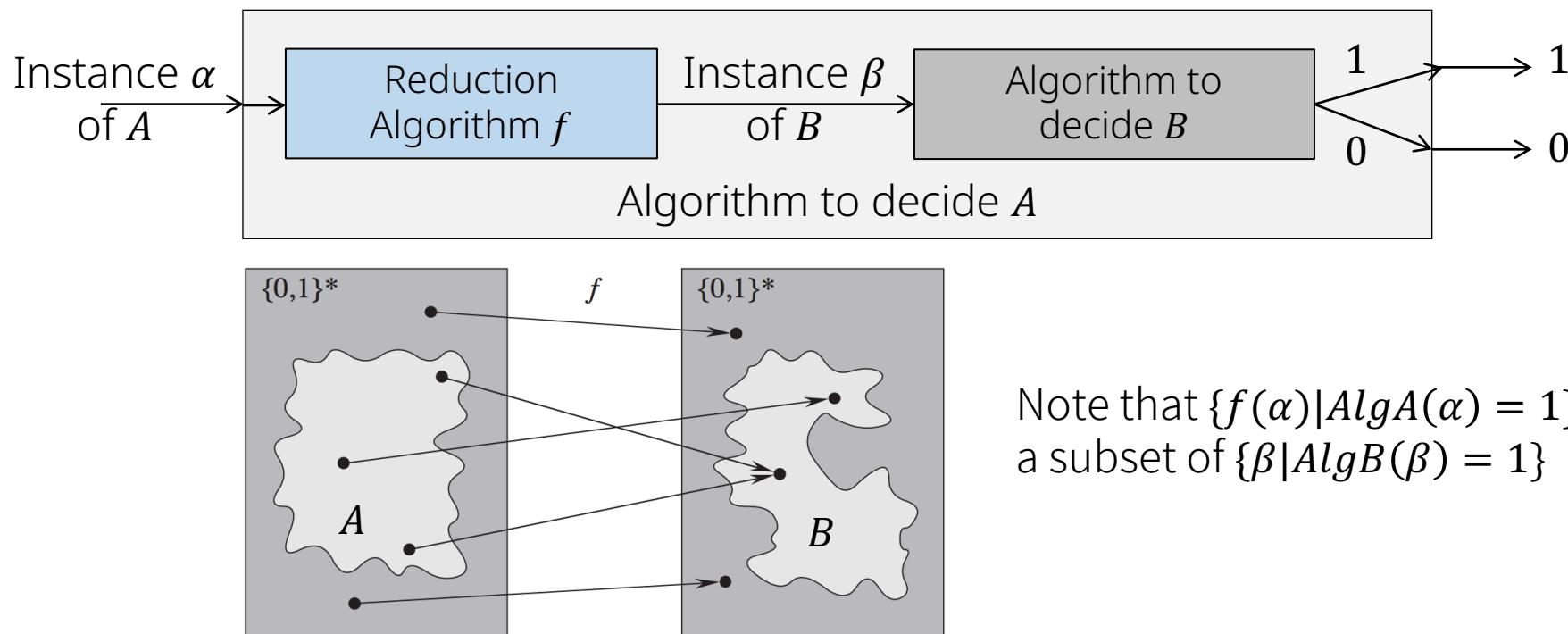
Reduction

- A **reduction** is an algorithm for **transforming every instance** of a problem A into an instance of another problem B
 - We focus on decision problems here
- A **polynomial-time reduction** can help determine the hardness relationship between problems
 - We write $A \leq_p B$ if A can be reduced to B in p-time
 - $A \leq_p B$ implies **A is no harder than B**

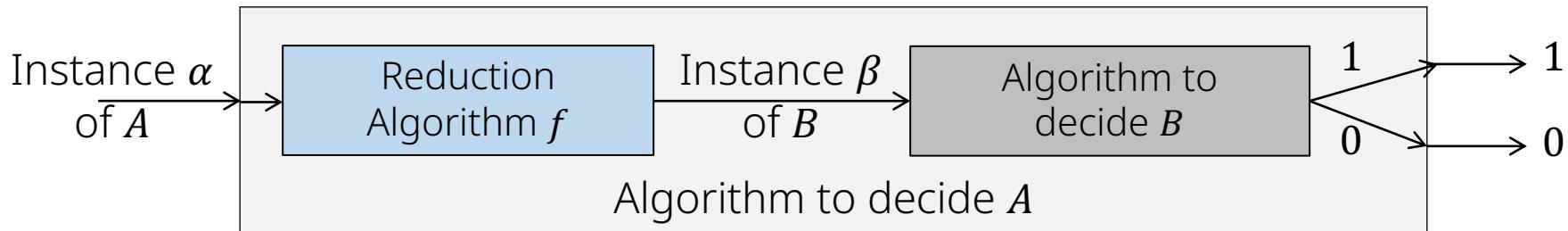


Reduction

- To show the correctness of a p-time reduction $A \leq_p B$:
 - Show that B outputs 1 **if and only if** A outputs 1
 - That is, for all α , $AlgA(\alpha) = 1$ iff $AlgB(f(\alpha)) = 1$
 - Show that the reduction algorithm is in polynomial time



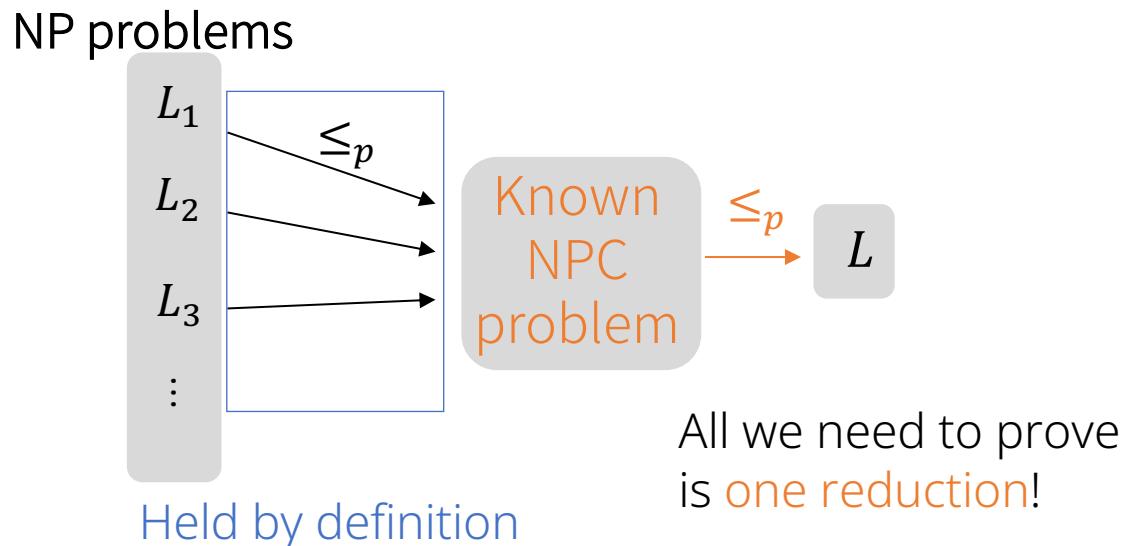
Applications of reduction: proving limits



- A common usage in NP-completeness proofs
 - Goal: prove that no known polynomial-time algorithm exists for B
 - Known: no known polynomial-time algorithm exists for A
 - Approach: construct a polynomial-time reduction algorithm to convert α to β
 - Why correct? 用反證法說明：若 B 有 p-time 解，則根據 reduction， A 也能在 p-time 解決。矛盾，故得証 B 無 p-time 解。

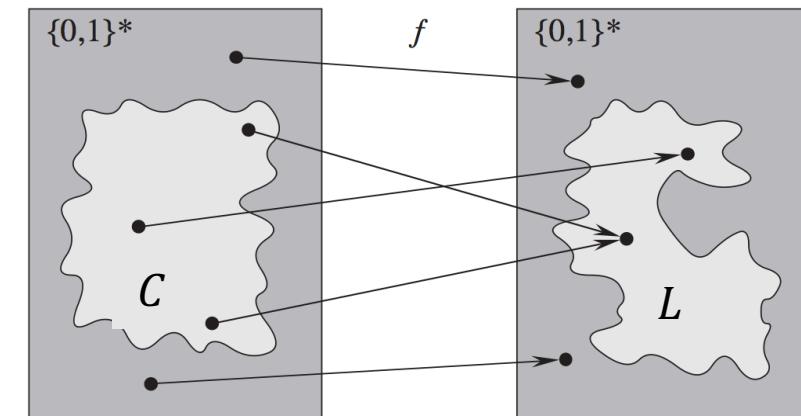
Proving NP-Completeness

- **Class 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. $L \in \text{NP}$
 2. $L \in \text{NP-Hard}$ (that is, $L' \leq_p L$ for every $L' \in \text{NP}$)



Proving NP-Completeness

- $L \in \text{NP-Complete} \Leftrightarrow L \in \text{NP} \text{ and } L \in \text{NP-hard}$
- Step-by-step approach for proving L in NPC:
 1. Prove $L \in \text{NP}$
 2. Prove $L \in \text{NP-hard } (C \leq_p L)$
 - 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 if and only if $f(x)$ in L for all x in $\{0,1\}^*$
 - ④ Prove that f is a polynomial time transformation



Polynomial-time Solving & Polynomial-time Verification

Chap. 34.1, 34.2

Abstract problems

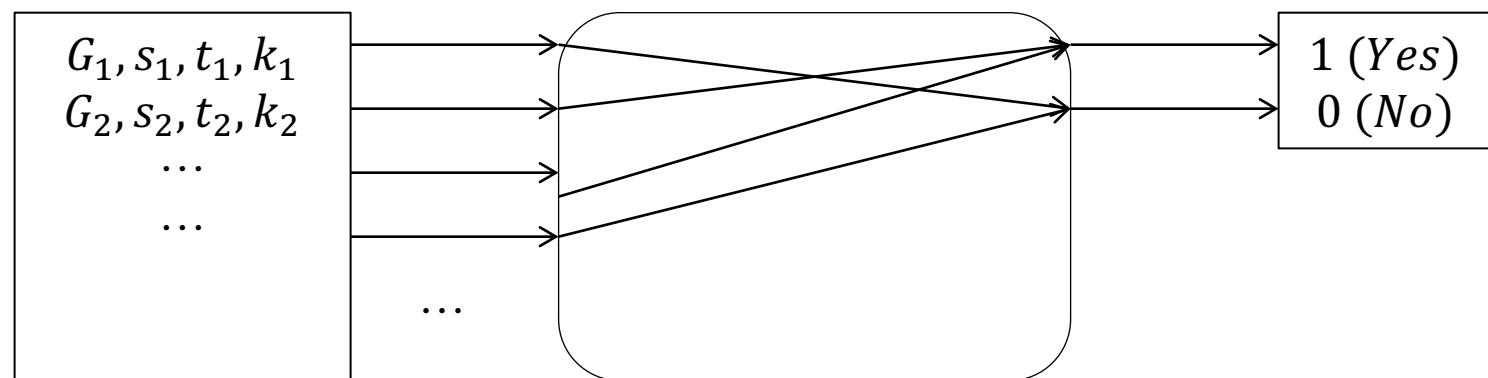
- I : the set of problem instances
 - S : the set of problem solutions
 - Q : an abstract problem, defined as a **binary relation** on I and S

Example of a decision problem, **SHORTEST-PATH**:

$I: < G, src, dest, k >$
所有可能的圖、起訖
點、長度上限

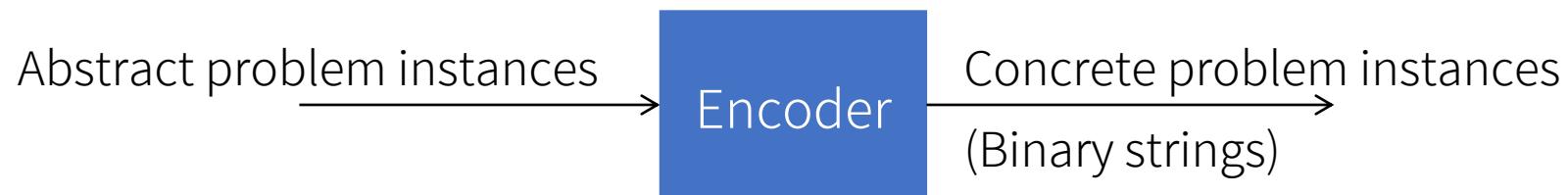
Q: SHORTEST-PATH

$S: \{0,1\}$

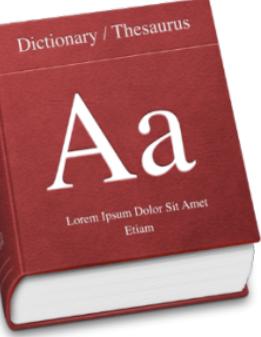


Encoding of problem instances

- Convert an abstract problem instance into a binary string that is 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



Formal-language theory

- An **Alphabet** Σ = a finite set of symbols
- A **Language** L over Σ = any set of strings made up from alphabet Σ
- Notations
 - ε : empty string
 - ϕ : empty language
 - Σ^* : the language of all strings over Σ

Q: In English, what is the alphabet Σ and the language L ?

- o $\Sigma = \{a, b, c, \dots, z\}$
- o L = a set of English words

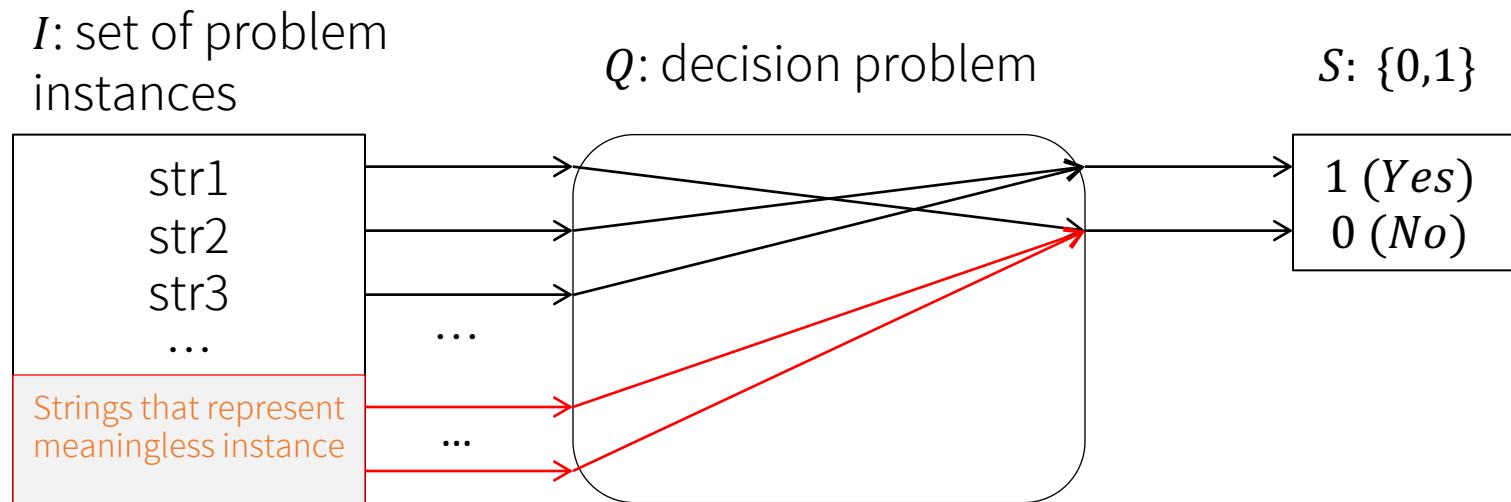
Q: In the language of binary representation of prime numbers, what is the alphabet Σ and the language L ?

- o $\Sigma = \{0,1\}$
- o $L = \{10, 11, 101, 111, \dots\}$

Q: If $\Sigma = \{0,1\}$, what is Σ^* ?

- o $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

Representation of a decision problem

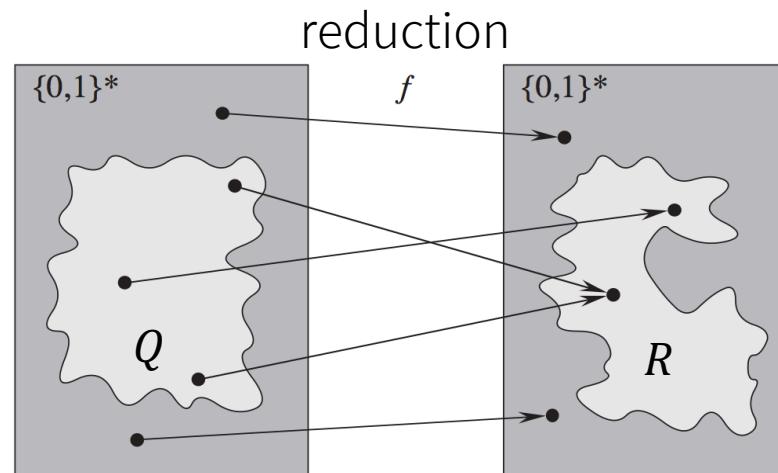


- $I = \text{the set of problem instances} = \Sigma^*$, where $\Sigma = \{0, 1\}$
- $Q = \text{a decision problem} = \text{a language } L \text{ over } \Sigma = \{0, 1\} \text{ such that } L = \{x \in \Sigma^*: Q(x) = 1\}$
 - 以答案為 1 的 instances 定義 decision problem Q !
 - $L = \{str2, str3\}$ in this simple example

Decision problem

A decision problem Q can be defined as a language L over $\Sigma = \{0, 1\}$ such that $L = \{x \in \Sigma^*: Q(x) = 1\}$

- 以答案為 1 的 instances 定義 decision problem Q
- Q can be reduced to R means $x \in Q \Leftrightarrow f(x) \in R$



Definition of the complexity class P in language-theoretic framework

- 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 means…
 - A accepts every string $x \in L$
 - 對所有在 L 中的 string 都要能正確地輸出 YES
 - 對不在 L 中的 string 可能輸出 NO, 也可能無法判斷 (e.g., loop forever)
- An algorithm A **decides** a language L if A accepts L and A rejects every string $x \notin L$
 - 對所有的string都要能正確地輸出YES/NO



Definition of the complexity class P in language-theoretic framework

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

Definition of Class P

$P = \{L \subseteq \{0,1\}^* \mid \text{there exists an algorithm that decides } L \text{ in p-time}\}$

Theorem 34.2

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

$P = \{L \subseteq \{0,1\}^* \mid L \text{ is accepted by a polynomial-time algorithm}\}$

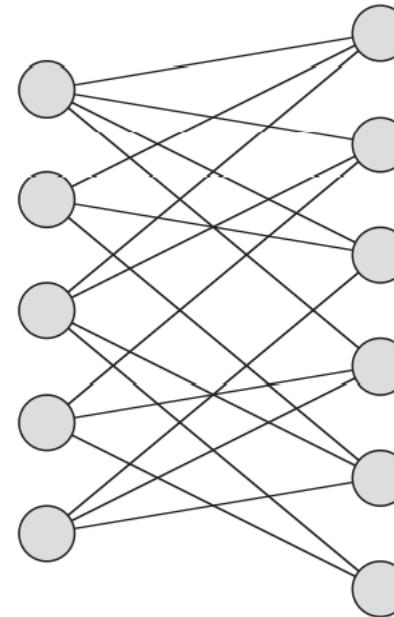
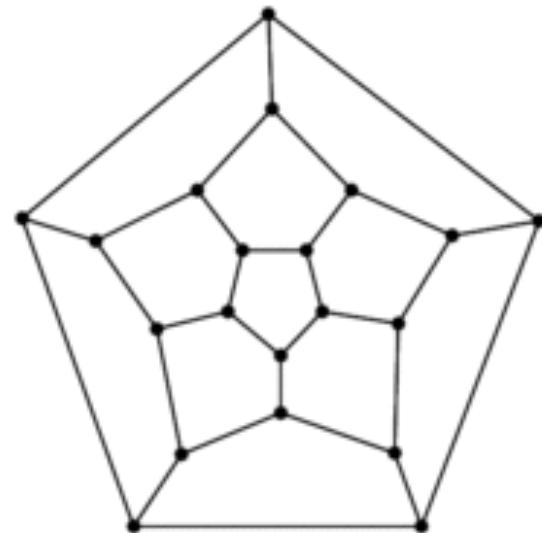
Hamiltonian-Cycle Problem

HAM–CYCLE = { $\langle G \rangle \mid G$ has a Hamiltonian cycle}

Q: Can you find a cycle visiting each vertex exactly once in the examples?

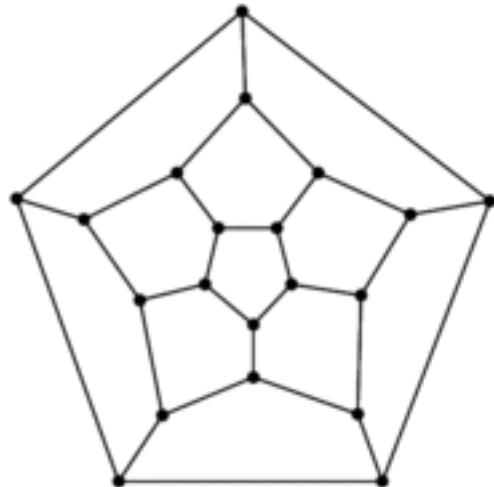
Q: Is this language decidable?

Q: Is this language decidable in polynomial time?



Polynomial-time Verification: Hamiltonian-cycle problem

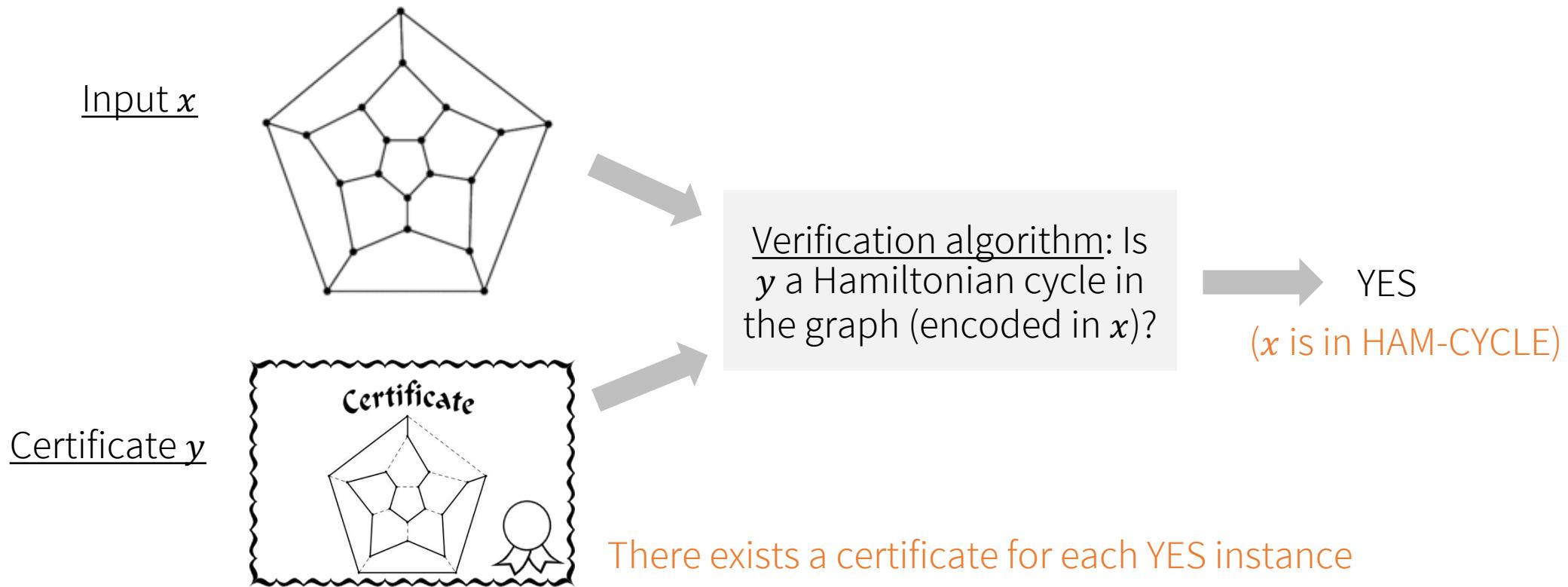
- Suppose a nice TA gives you a **certificate** – a vertex sequence that forms a Hamiltonian cycle in a given graph G
- With this certificate, how much time does it take to **verify** that G indeed contains a Hamiltonian cycle?



Verification algorithms

- Verification algorithms verify memberships in language

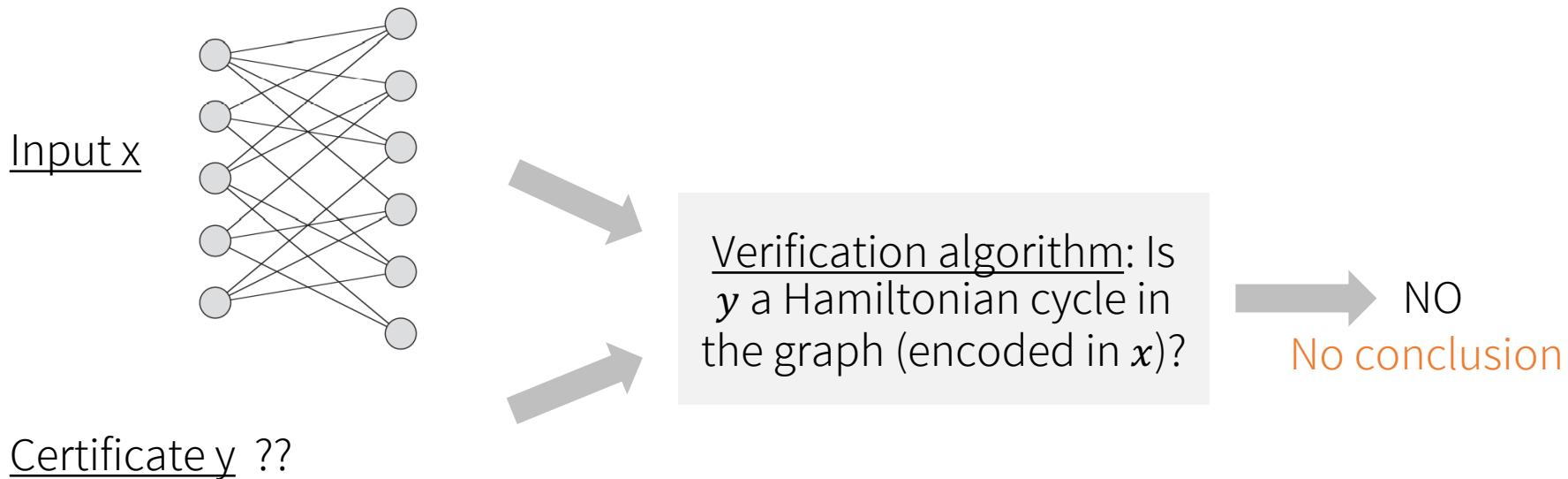
$$\text{HAM-CYCLE} = \{\langle G \rangle \mid G \text{ has a Hamiltonian cycle}\}$$



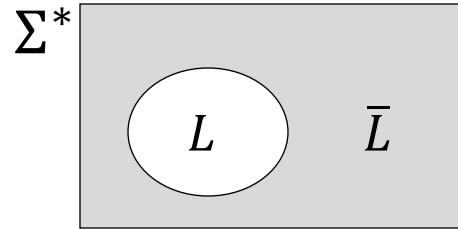
Verification algorithms

- Verification algorithms verify memberships in language

$$\text{HAM-CYCLE} = \{\langle G \rangle \mid G \text{ has a Hamiltonian cycle}\}$$



Co-NP

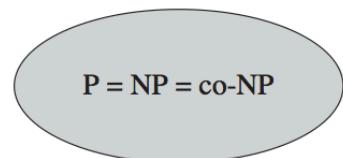


- Class **co-NP**: class of decision problems whose **complement** are in NP
 - Complement of $L = \bar{L} = \Sigma^* - L$
- **HAM–CYCLE**: Given a graph G , is there a simple cycle that visits each vertex on G exactly once?
- **$\overline{\text{HAM–CYCLE}}$** : Given a graph G , does G contain no Hamiltonian cycle?
- It's still an open problem whether **HAM–CYCLE** ∈ co-NP
 - Equivalently, whether **$\overline{\text{HAM–CYCLE}}$** ∈ NP
 - 是否能提供 certificate 證明 graph G 沒有 Hamiltonian cycle?

P, NP, and co-NP

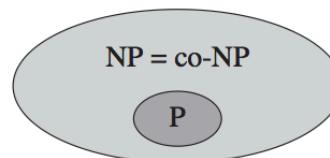
- Class P: class of decision problems **solvable** in $O(n^k)$
 - $O(n^k)$ means polynomial in the input size (k is a constant)
- Class NP: class of decision problems **verifiable** in $O(n^k)$
- Class co-NP: class of decision problems whose **complement** are in NP

If $P = NP = \text{co-NP}$



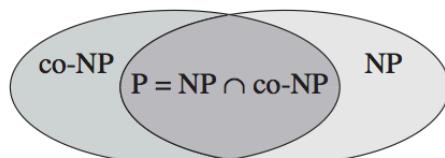
(a)

If $P \neq NP$ and $NP = \text{co-NP}$



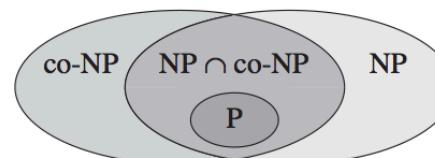
(b)

If $P = NP \cap \text{co-NP}$



(c)

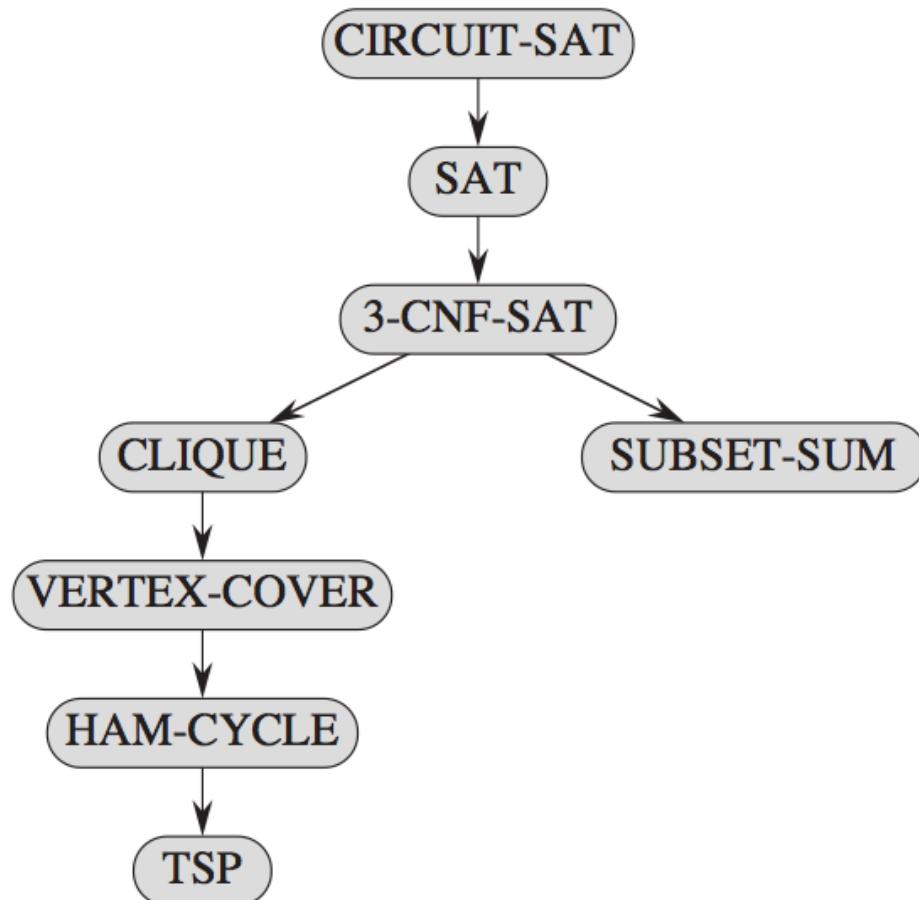
Most likely case!



(d)

Classic NP-Complete Problems

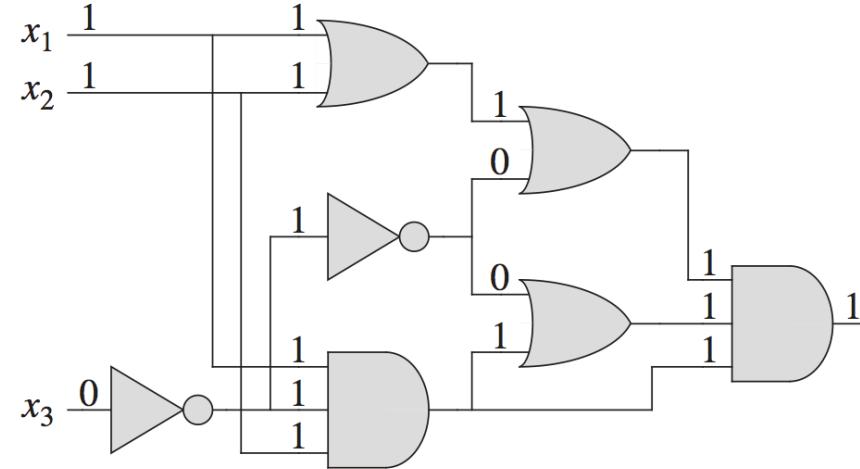
Roadmap



$$A \rightarrow B : A \leq_p B$$

Circuit-Satisfiability Problem

CIRCUIT-SAT = { $\langle C \rangle$: C is a satisfiable Boolean combinational circuit}



Cook's Theorem

The Circuit-Satisfiability Problem (CIRCUIT-SAT) is NP-complete

Formula satisfiability problem (SAT)

$SAT = \{\Phi \mid \Phi \text{ is a Boolean Formula with a satisfying assignment}\}$

- Given a Boolean formula ϕ , is there a variable assignment satisfying ϕ ?
 - \wedge (AND), \vee (OR), \neg (NOT), \rightarrow (implication), \leftrightarrow (if and only if)
 - Satisfiable = Φ is evaluated to 1
- Example: $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
 - $<0, 0, 1, 1>$ is one solution

Formula satisfiability problem (SAT)

$SAT = \{\Phi \mid \Phi \text{ is a Boolean Formula with a satisfying assignment}\}$

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

SAT \in NP

- **Polynomial-time verification:** replaces each variable in the formula with the corresponding value in the **certificate** and then evaluates the expression
- Example: $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

$$x_1 = 0$$

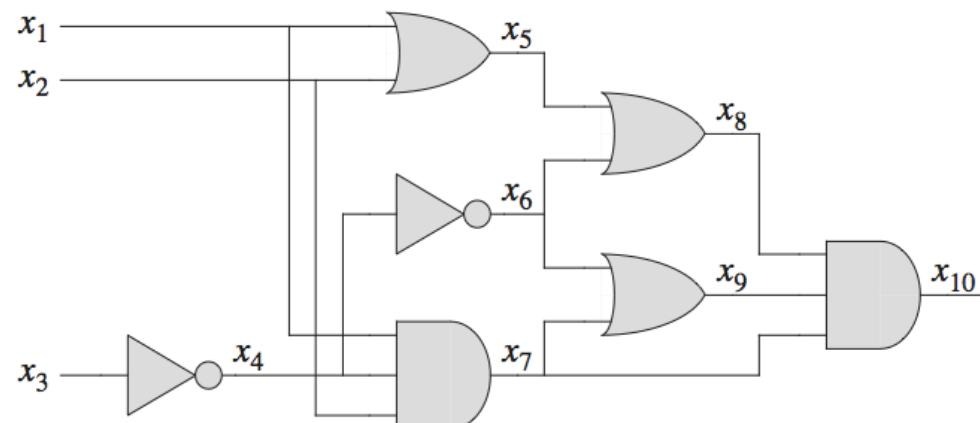
$$x_2 = 0$$

$$x_3 = 1$$

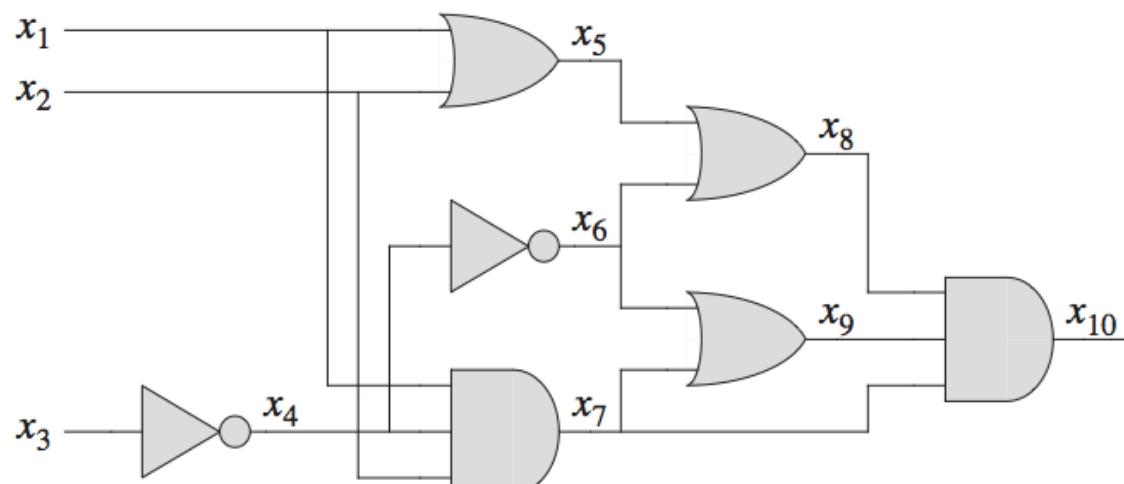
$$x_4 = 1$$

SAT \in NP-hard

- ① CIRCUIT–SAT is a known NPC problem
- ② Construct a reduction f transforming every instance of CIRCUIT–SAT to an instance of SAT
 - Assign a variable to each **wire** in circuit C
 - Represent the operation of each gate using a formula, e.g., $x_{10} \leftrightarrow x_7 \wedge x_8 \wedge x_9$
 - $\phi = \text{AND the output variable and the operations of all gates}$



SAT \in NP-hard



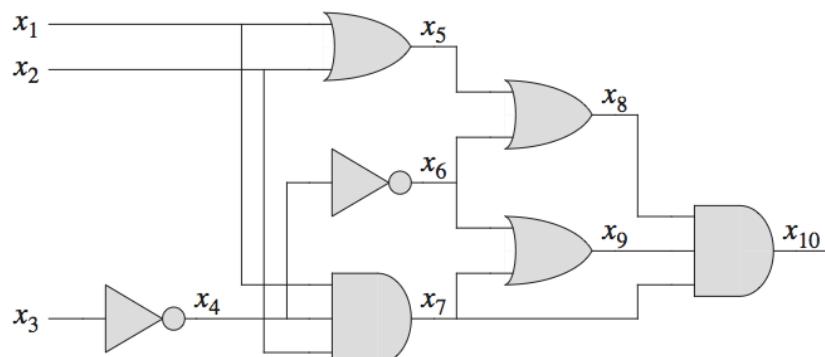
$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))\end{aligned}$$

SAT \in NP-hard

③ Prove that $x \in \text{CIRCUIT-SAT} \Leftrightarrow f(x) \in \text{SAT}$

- $x \in \text{CIRCUIT-SAT} \Rightarrow f(x) \in \text{SAT}$
- $f(x) \in \text{SAT} \Rightarrow x \in \text{CIRCUIT-SAT}$

④ f is a polynomial time transformation



$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))\end{aligned}$$

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

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

- 3-conjunctive normal form (3-CNF)
 - 3-CNF = AND of clauses, each is the OR of exactly 3 distinct literals
 - A literal is an occurrence of a variable or its negation, e.g., x_1 or $\neg x_1$
- Example: $(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$
- $<0, 0, 1, 1>$ is one solution

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

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

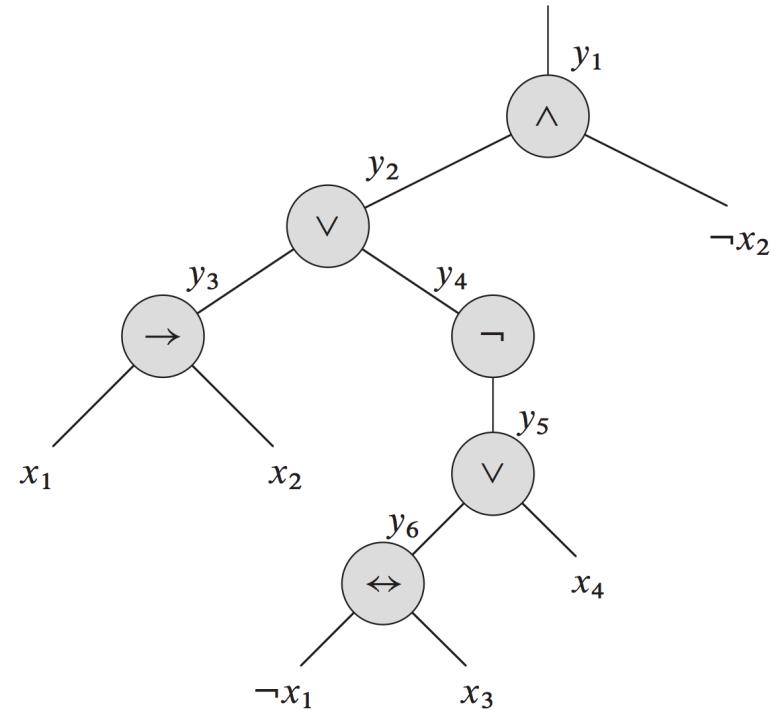
- Is 3-CNF-SAT \in NP-Complete?
- To prove 3-CNF-SAT is NP-Complete, we show that
 - 1. 3-CNF-SAT \in NP
 - 2. 3-CNF-SAT \in NP-hard ($SAT \leq_p 3-CNF-SAT$)
 - ① SAT is a known NPC problem
 - ② Construct a reduction f transforming every instance of SAT to an instance of 3-CNF-SAT
 - ③ Prove that $x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

* We will focus on the construction of reduction functions from now on; but a full proof requires showing the other conditions are true as well

$\text{SAT} \leq_p \text{3-CNF-SAT}$

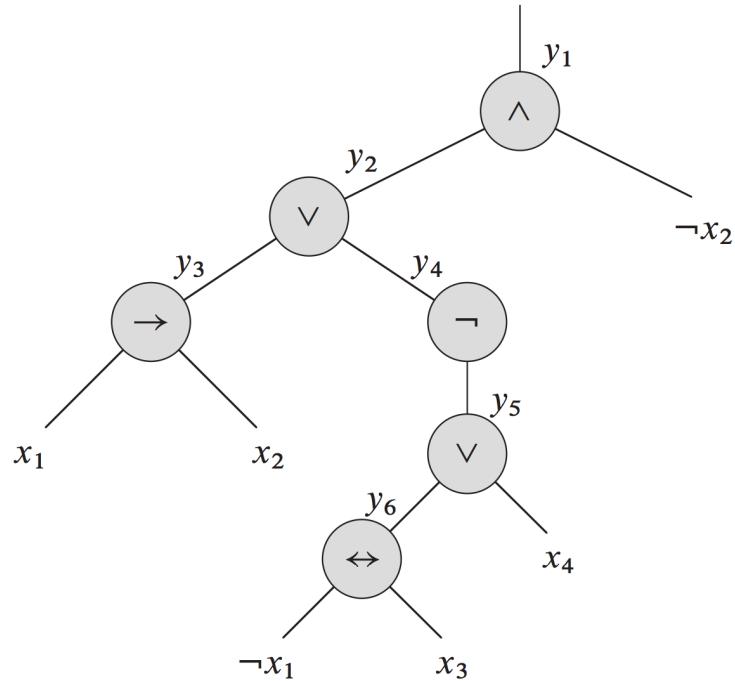
- a) Construct a **binary parser tree** for input formula ϕ and introduce a variable y_i for the output of each internal node

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



$\text{SAT} \leq_p \text{3-CNF-SAT}$

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



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

ϕ'_1

$\text{SAT} \leq_p \text{3-CNF-SAT}$

c) Convert each clause ϕ'_i to CNF

- o Construct a truth table for each clause ϕ'_i
- o Construct the disjunctive normal form for $\neg\phi'_i$
- o Apply DeMorgan's Law to get the CNF formula ϕ''_i

$$\begin{aligned}\neg(a \wedge b) &= \neg a \vee \neg b \\ \neg(a \vee b) &= \neg a \wedge \neg b\end{aligned}$$

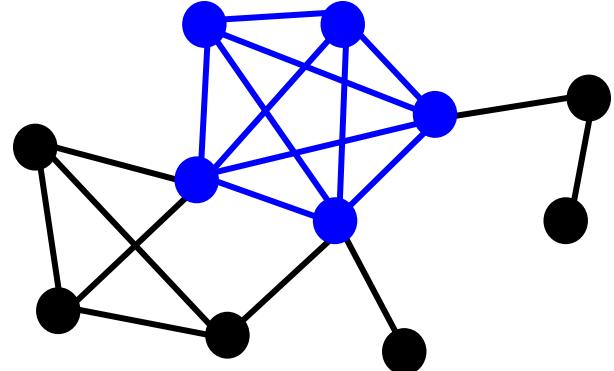
y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$	ϕ'_1	$\neg\phi'_1$	$\neg\phi'_1 =$	$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$
1	1	1	0		1	$\phi''_1 =$	$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2)$
1	1	0	1		0		$\wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$
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		

$\text{SAT} \leq_p \text{3-CNF-SAT}$

- d) Construct ϕ''' s.t. each clause C_i has **exactly 3 distinct literals**
- C_i has 3 distinct literals: do nothing
 - C_i has 2 distinct literals: $C_i = I_1 \vee I_2 = (I_1 \vee I_2 \vee p) \wedge (I_1 \vee I_2 \vee \neg p)$
 - C_i has 1 literal: $C_i = I = (I \vee p \vee q) \wedge (I \vee \neg p \vee q) \wedge (I \vee p \vee \neg q) \wedge (I \vee \neg p \vee \neg q)$
 - Show that ϕ''' is satisfiable iff ϕ is satisfiable
 - Show that all transformation can be done in polynomial time
 - $\Rightarrow \text{3-CNF-SAT}$ is NP-Complete

The 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 clique of maximum size in G
- Decision problem: find a clique of size k in G



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

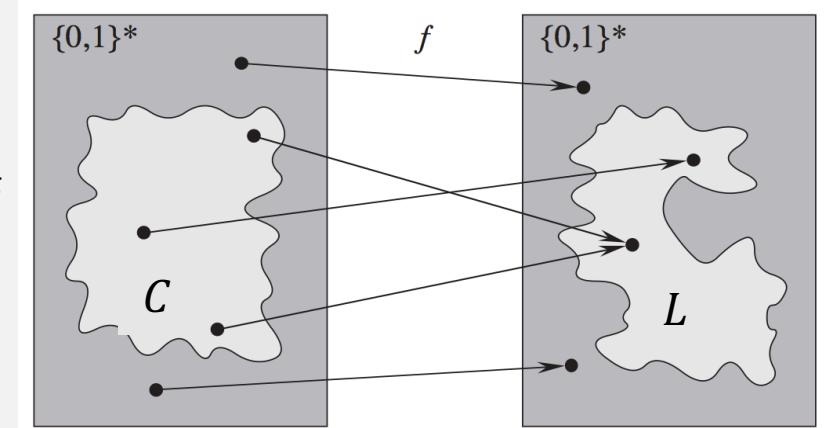
The Clique Problem

CLIQUE = { $\langle G, k \rangle$: G is a graph containing a clique of size k }

- Prove that **CLIQUE** \in NP-COMPLETE
- Key idea: 3-CNF-SAT \leq_p CLIQUE

Step-by-step approach for proving L in NPC:

- Prove $L \in \text{NP}$
- Prove $L \in \text{NP-hard}$ ($C \leq_p L$)
 - Select a known NPC problem C
 - Construct a reduction f transforming every instance of C to an instance of L
 - Prove that x in C if and only if $f(x)$ in L for all x in $\{0,1\}^*$
 - Prove that f is a polynomial time transformation



The Clique Problem

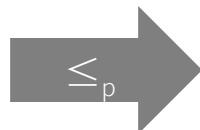
CLIQUE = { $\langle G, k \rangle$: G is a graph containing a clique of size k }

- Prove that CLIQUE \in NP-COMPLETE
- Key idea: 3-CNF-SAT \leq_p CLIQUE
 - Construct a graph G s.t. ϕ with k clauses is satisfiable $\Leftrightarrow G$ has a clique of size k

Example: $\phi = 1 \Leftrightarrow G$ has a clique of size 3

Satisfying assignment $\langle x_1, x_2, x_3 \rangle = \langle X, 0, 1 \rangle$

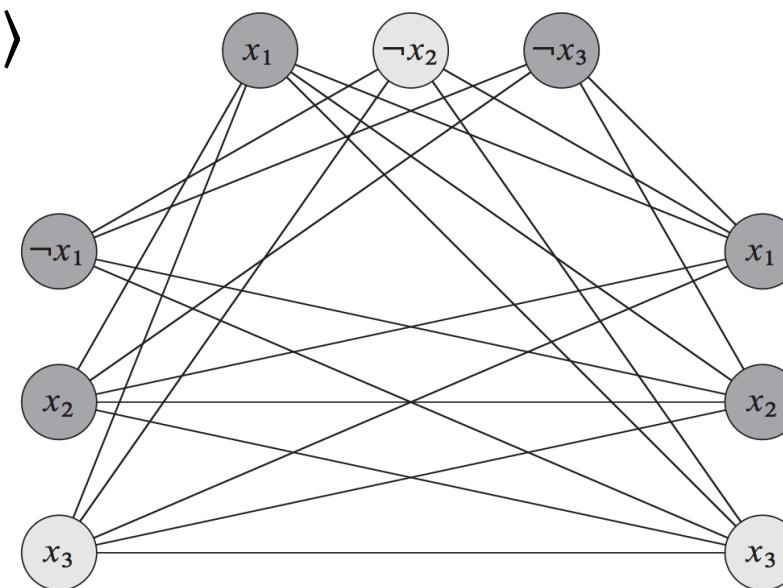
$$\begin{aligned}\phi = & (x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (\neg x_1 \vee x_2 \vee x_3) \\ & \wedge (x_1 \vee x_2 \vee x_3)\end{aligned}$$



$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$

$$C_2 = \neg x_1 \vee x_2 \vee x_3$$

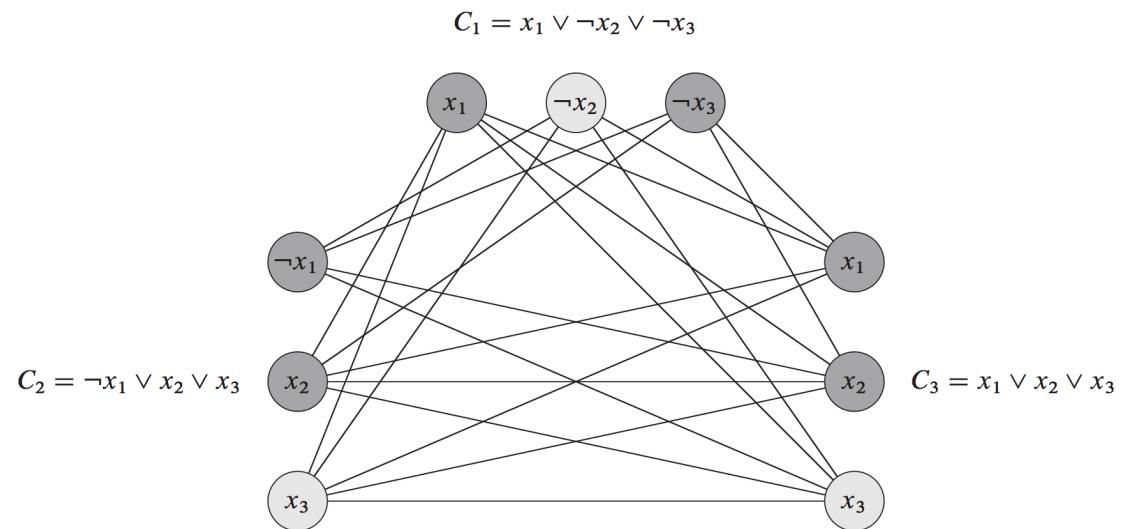
$$C_3 = x_1 \vee x_2 \vee x_3$$



3-CNF-SAT \leq_p CLIQUE

Polynomial-time reduction:

- Let $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$ 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 $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ , 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, and
 - l_i^r is not the negation of l_j^s



3-CNF-SAT \leq_p CLIQUE

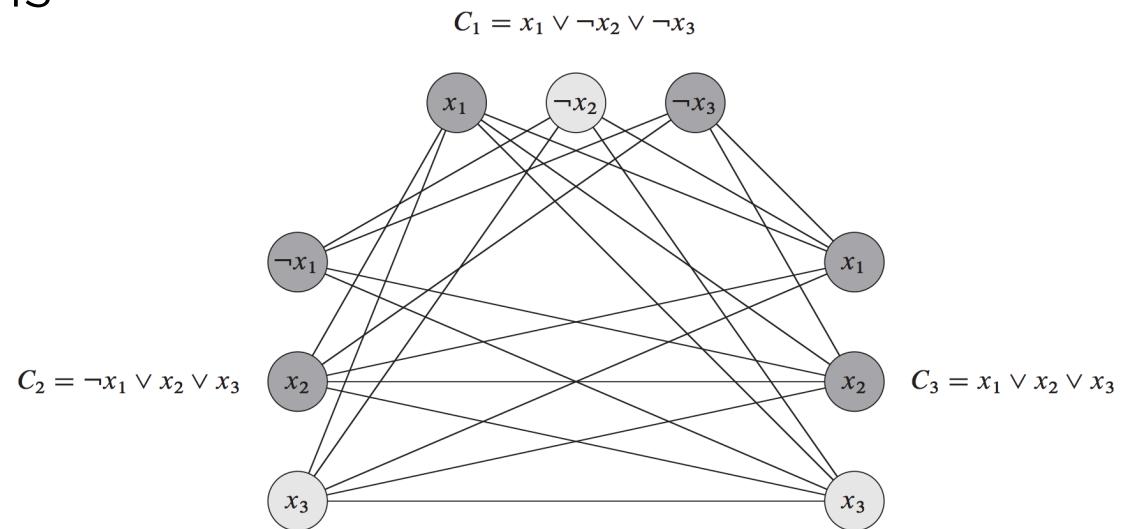
Correctness proof: ϕ is satisfiable $\Rightarrow G$ has a clique of size k

If ϕ is satisfiable,

\Rightarrow Each C_r contains at least one $l_i^r = 1$ and each such literal corresponds to a vertex v_i^r

\Rightarrow Picking a *TRUE* literal from each C_r forms a set of V' of k vertices

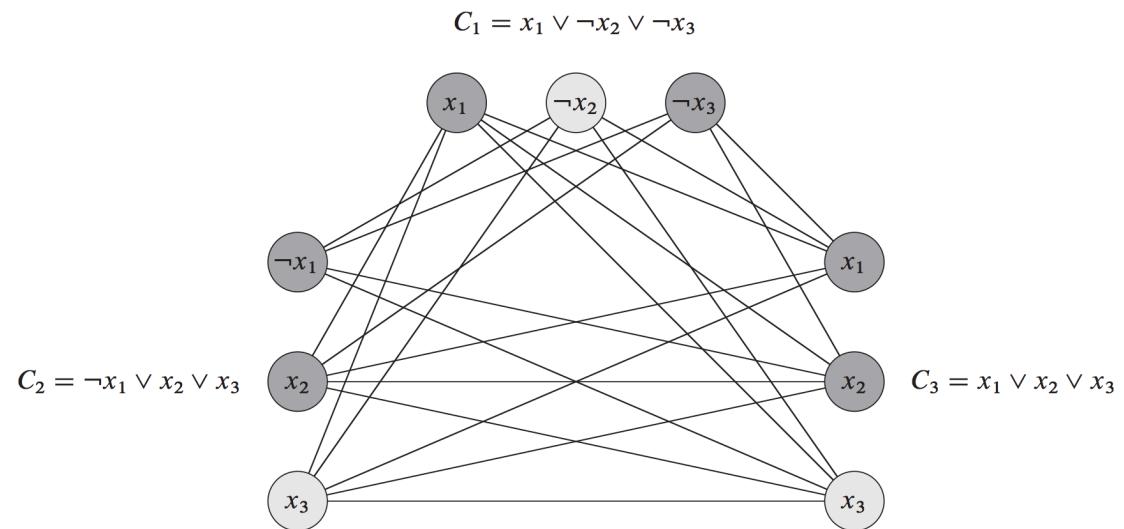
\Rightarrow For any two vertices $v_i^r, v_j^s \in V'$, edge $(v_i^r, v_j^s) \in E$, because $r \neq s$ and $l_i^r = l_j^s = 1$ (l_i^r, l_j^s cannot be complements)



$3\text{-CNF-SAT} \leq_p \text{CLIQUE}$

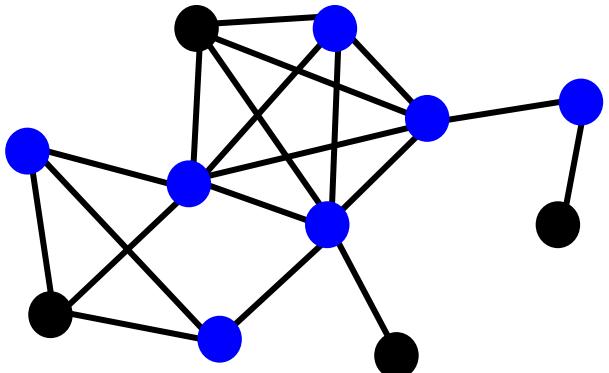
Correctness proof: G has a clique of size $k \Rightarrow \phi$ is satisfiable

- If G has a clique V' of size k
- $\Rightarrow V'$ contains exactly one vertex per triple since no edges connect vertices in the same triple
- \Rightarrow Assign *TRUE* to each l_i^r where $v_i^r \in V'$ Q: why is this a feasible assignment?
- \Rightarrow each C_r is satisfied, and so is ϕ



The vertex-cover problem

- A vertex cover of $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(w, v) \in E$, then $w \in V'$ or $v \in V'$ or both
 - A vertex cover “covers” every edge in G
- Optimization problem: find a vertex cover of minimum size in G
- Decision problem: find a vertex cover of size k in G



Does G have a vertex cover of size 11?
Does G have a vertex cover of size 7?
Does G have a vertex cover of size 6?

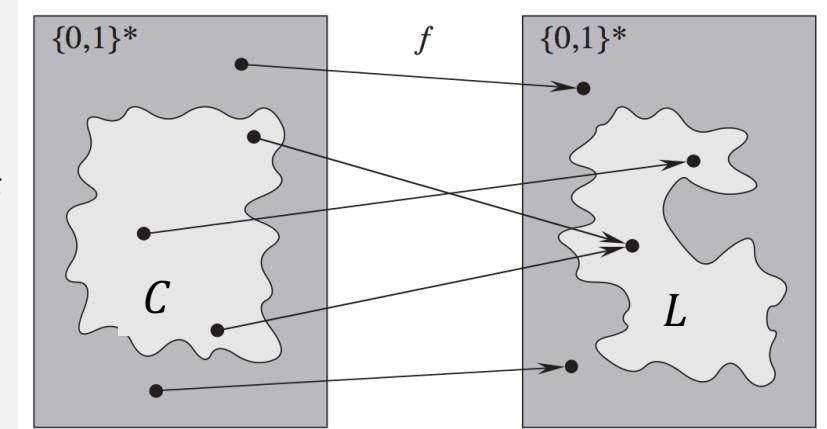
The Vertex-Cover Problem

VERTEX-COVER = $\{(G, k): \text{graph } G \text{ has a vertex cover of size } k\}$

- Prove that VERTEX-COVER \in NP-Complete
- Polynomial-time reduction: CLIQUE \leq_p VERTEX-COVER

Step-by-step approach for proving L in NPC:

- Prove $L \in \text{NP}$
- Prove $L \in \text{NP-hard}$ ($C \leq_p L$)
 - Select a known NPC problem C
 - Construct a reduction f transforming every instance of C to an instance of L
 - Prove that x in C if and only if $f(x)$ in L for all x in $\{0,1\}^*$
 - Prove that f is a polynomial time transformation

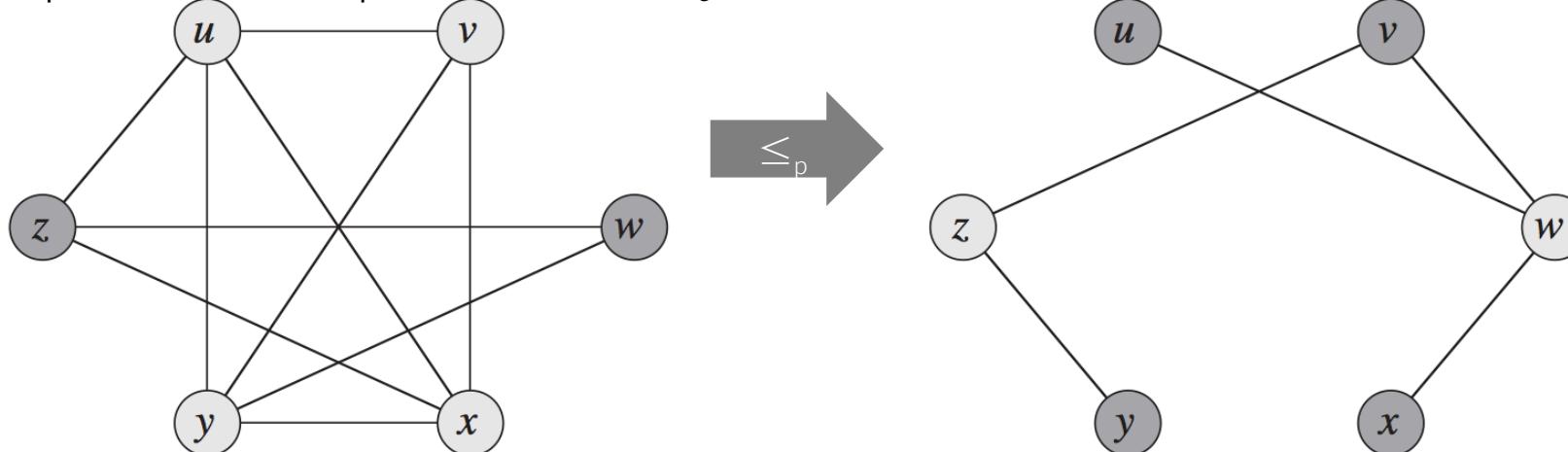


The Vertex-Cover Problem

VERTEX-COVER = $\{\langle G, k \rangle : \text{graph } G \text{ has a vertex cover of size } k\}$

- Prove that VERTEX-COVER \in NP-Complete
- Polynomial-time reduction: CLIQUE \leq_p VERTEX-COVER
 - Compute the complement of G
 - Given $G = \langle V, E \rangle$, G_c is defined as $\langle V, E_c \rangle$ s.t. $E_c = \{(u, v) | (u, v) \notin E\}$
 - We want to prove that G has a clique of size $k \Leftrightarrow G$'s complement (G_c) has a vertex cover of size $|V| - k$

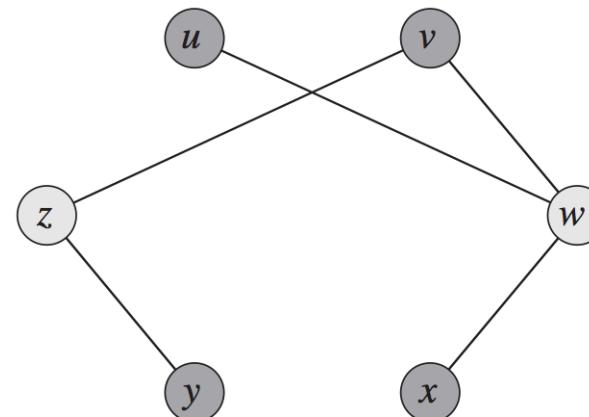
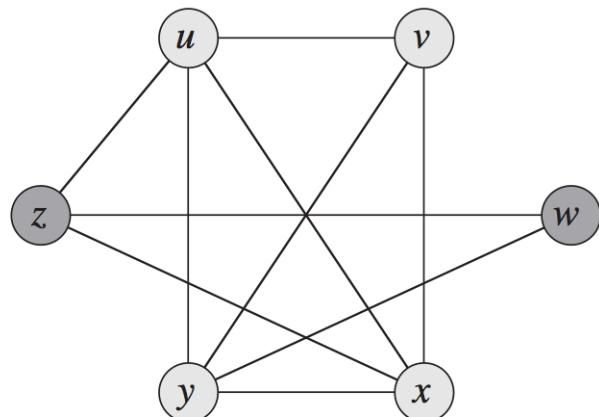
Example: G has a clique of size 4 $\Leftrightarrow G_c$ has a vertex cover of size 2



CLIQUE \leq_p VERTEX-COVER

Correctness proof: G has a clique of size $k \Rightarrow G_c$ has as a vertex cover of size $\underline{|V| - k}$

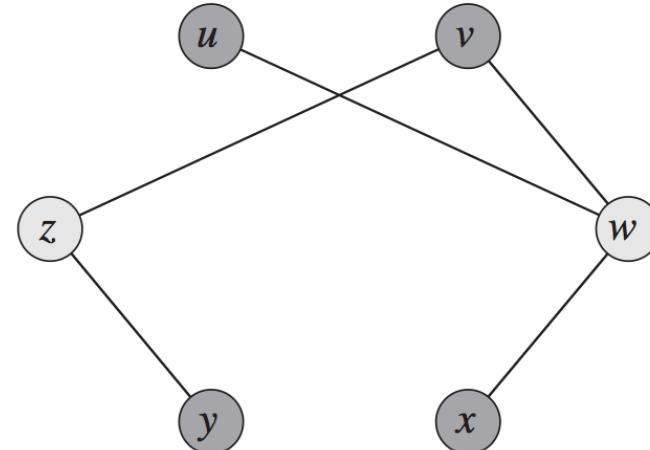
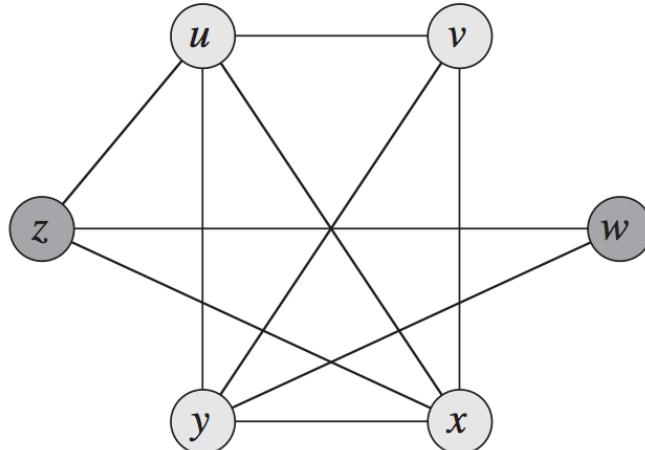
- Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$
- \Rightarrow for all $(w, v) \in E_c$, at least one of w or $v \notin V'$
- $\Rightarrow w \in V - V'$ or $v \in V - V'$ (or both)
- \Rightarrow edge (w, v) is covered by $V - V'$
- $\Rightarrow V - V'$ forms a vertex cover of G_c , and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



$\text{CLIQUE} \leq_p \text{VERTEX-COVER}$

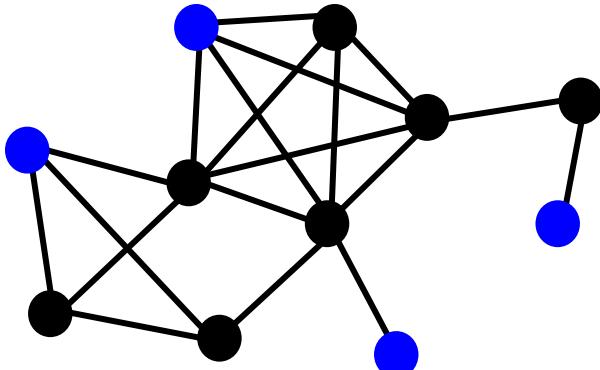
Correctness proof: G_c has as a vertex cover of size $|V| - k \Rightarrow G$ has a clique of size k

- Suppose G_c has a vertex cover $V' \subseteq V$ with $|V'| = |V| - k$
- $\Rightarrow \forall a, b \in V$, if $(a, b) \in E_c$, then $a \in V'$ or $b \in V'$ or both
- $\Rightarrow \forall a, b \in V$, if $a \notin V'$ and $b \notin V'$, then $(a, b) \notin E_c$; that is, $(a, b) \in E$
- $\Rightarrow V - V'$ is a clique, and $|V - V'| = k$



The independent-set problem

- An independent set of $G = (V, E)$ is a subset $V' \subseteq V$ such that G has no edge between any pair of vertices in V'
- Optimization problem: find an independent set of maximum size in G
- Decision problem: find an independent set of size k in G

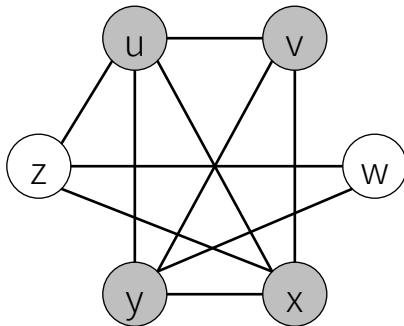


Does G have an independent set of size 1?
Does G have an independent set of size 4?
Does G have an independent set of size 5?

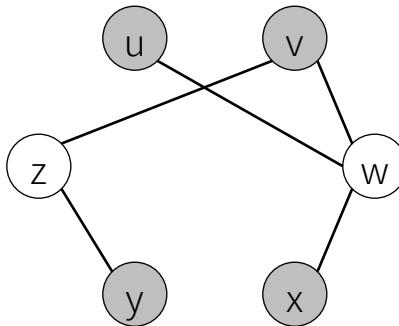
Practice: show that IND-SET is NP-Complete (Textbook Problem 34-1)

Clique, Independent-Set, Vertex-Cover

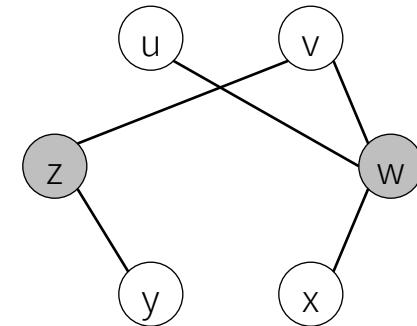
- The following are equivalent for $G = (V, E)$ and a subset V' of V :
 1. V' is a clique of G
 2. V' is an independent set of G_c
 3. $V - V'$ is a vertex cover of G_c



Clique
 $V' = \{u, v, x, y\}$ in G



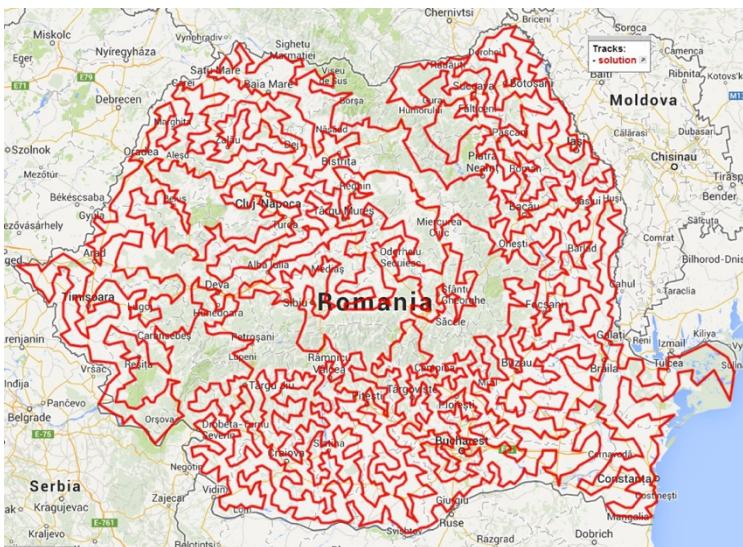
Independent set
 $V' = \{u, v, x, y\}$ in G_c



Vertex cover
 $V - V' = \{z, w\}$ 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: Given a set of cities and their pairwise distances, find a tour of cost at most k that visits each city exactly once.



Romanian TSP: 2950 nodes
<http://cadredidactice.ub.ro/ceraselacrisan/cercetare/>

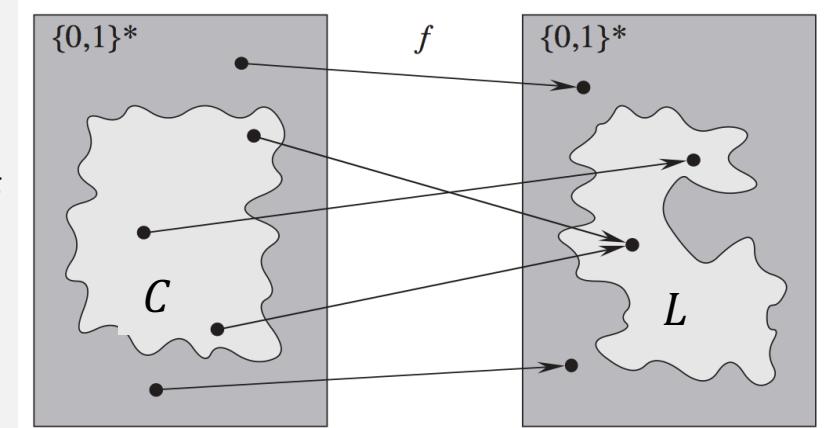
The TSP Problem

$TSP = \{\langle G, c, k \rangle : G = (V, E) \text{ is a complete graph, } c \text{ is a non-negative cost function for edges, } G \text{ has a traveling-salesman tour with cost at most } k\}$

- Prove that $\text{TSP} \in \text{NP-COMPLETE}$
- Polynomial-time reduction: $\text{HAM-CYCLE} \leq_p \text{TSP}$

Step-by-step approach for proving L in NPC:

- Prove $L \in \text{NP}$
- Prove $L \in \text{NP-hard}$ ($C \leq_p L$)
 - Select a known NPC problem C
 - Construct a reduction f transforming every instance of C to an instance of L
 - Prove that x in C if and only if $f(x)$ in L for all x in $\{0,1\}^*$
 - Prove that f is a polynomial time transformation

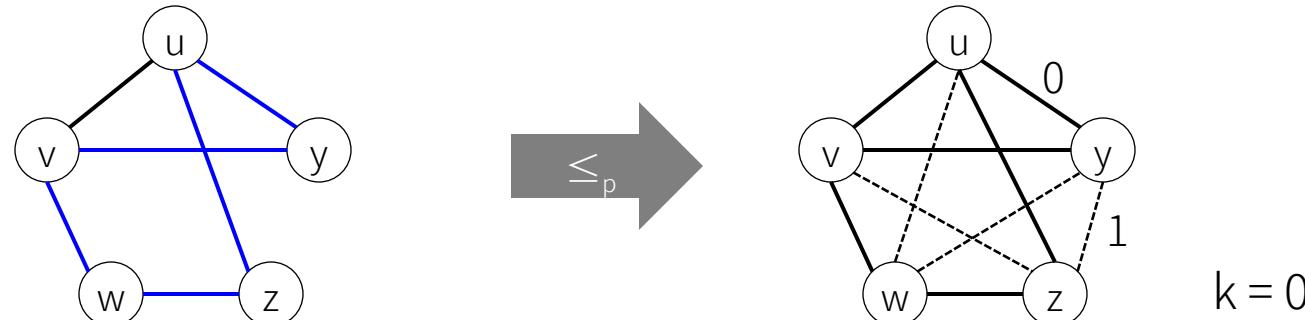


The TSP Problem

$TSP = \{\langle G, c, k \rangle : G = (V, E) \text{ is a complete graph, } c \text{ is a non-negative cost function for edges, } G \text{ has a traveling-salesman tour with cost at most } k\}$

- Prove that $TSP \in \text{NP-COMPLETE}$
- Polynomial-time reduction: HAM-CYCLE \leq_p TSP
 - Given a HAM-CYCLE instance, we construct a TSP instance in which G is a complete graph and

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

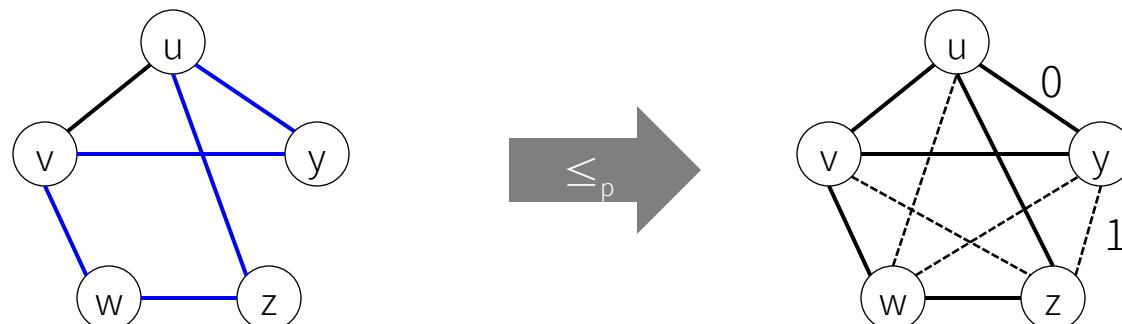


HAM-CYCLE \leq_p TSP

Correctness proof: $x \in \text{HAM-CYCLE} \Leftrightarrow f(x) \in \text{TSP}$

- More specifically, we want to prove that G contains a Hamiltonian cycle $h = \langle v_1, v_2, \dots, v_n, v_1 \rangle$ if and only if $\langle v_1, v_2, \dots, v_n, v_1 \rangle$ is a traveling-salesman tour with cost 0

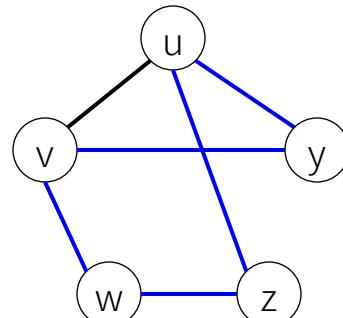
u, y, v, w, z, u is a Hamiltonian cycle $\Leftrightarrow u, y, v, w, z, u$ is a traveling-salesman tour with cost 0



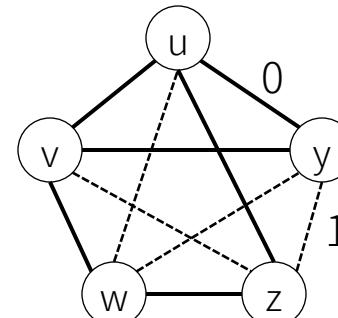
HAM–CYCLE \leq_p TSP

Correctness proof: $x \in \text{HAM–CYCLE} \Rightarrow f(x) \in \text{TSP}$

- Suppose the Hamiltonian cycle is $h = \langle v_1, v_2, \dots, v_n, v_1 \rangle$
- $\Rightarrow h$ is also a tour in the transformed TSP instance
- \Rightarrow The distance of the tour h is 0 since there are n consecutive edges in E , and so has distance 0 in $f(x)$
- $\Rightarrow f(x) \in \text{TSP}$ ($f(x)$ has a TSP tour with cost ≤ 0)



\leq_p



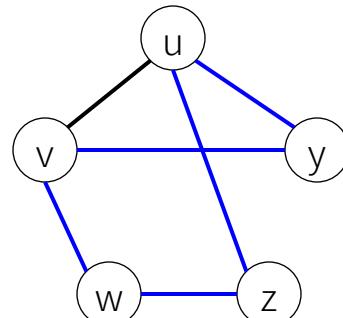
u, y, v, w, z, u is a Hamiltonian cycle

u, y, v, w, z, u is a traveling-salesman tour with cost 0

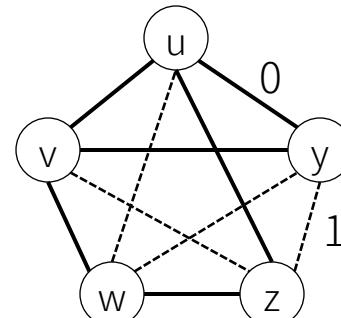
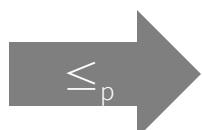
HAM-CYCLE \leq_p TSP

Correctness proof: $f(x) \in \text{TSP} \Rightarrow x \in \text{HAM-CYCLE}$

- Suppose **after reduction**, there is a TSP tour with cost ≤ 0 . Let it be $\langle v_1, v_2, \dots, v_n, v_1 \rangle$
- \Rightarrow The tour contains only edges in E
- \Rightarrow Thus, $\langle v_1, v_2, \dots, v_n, v_1 \rangle$ is a Hamiltonian cycle ($x \in \text{HAM-CYCLE}$).



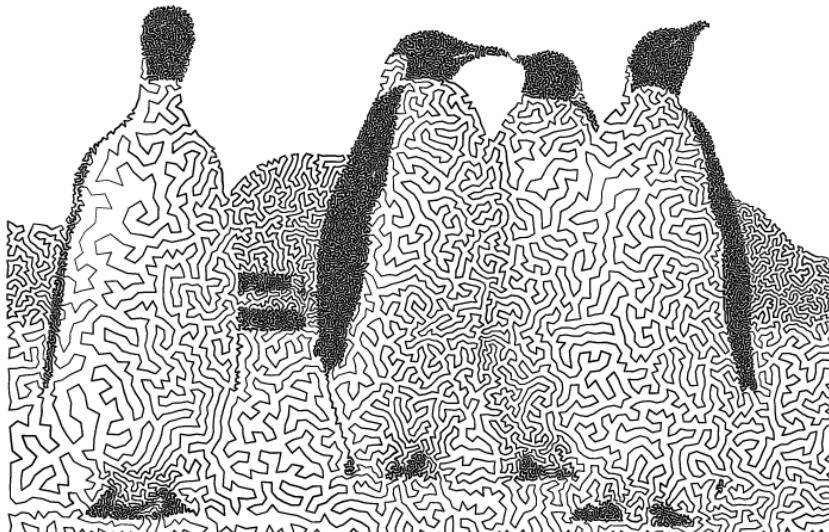
u, y, v, w, z, u is a Hamiltonian cycle



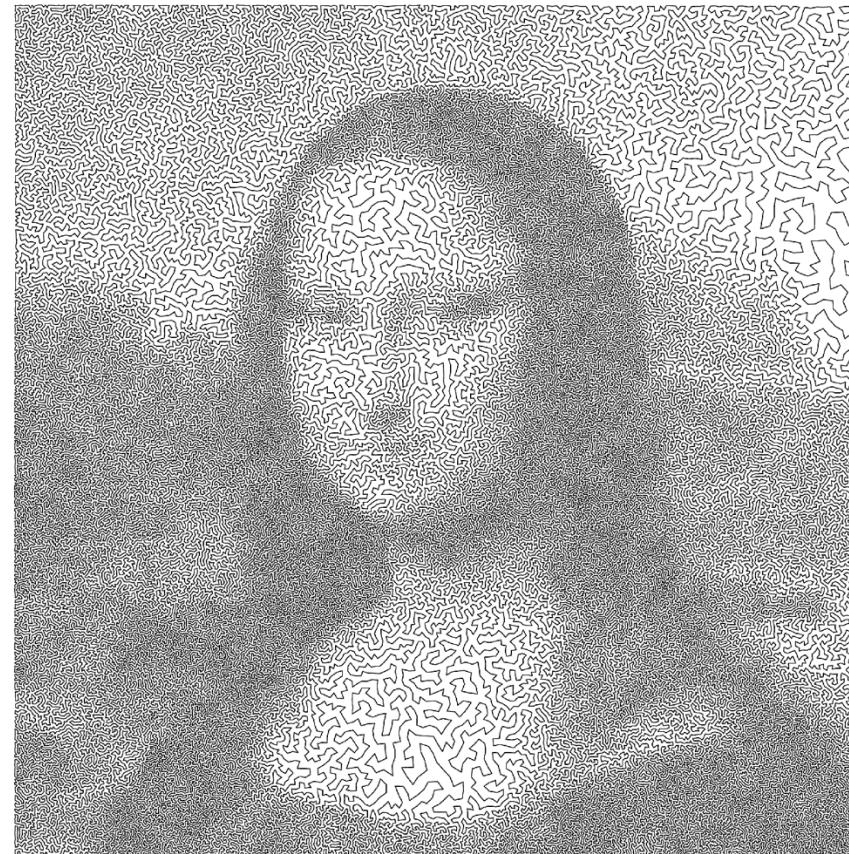
u, y, v, w, z, u is a traveling-salesman tour with cost 0

TSP arts and challenges

Mona Lisa TSP: \$1,000 Prize for a
100,000-city challenge problem



<http://www.cgl.uwaterloo.ca/~csk/projects/tsp/>



<http://www.math.uwaterloo.ca/tsp/>

Coping with NP-Hard problems

- Since polynomial-time solutions are unlikely (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**: make use of a randomizer (random number generator) for operation

Coping with NP-Hard problems

- Since polynomial-time solutions are unlikely (unless $P = NP$), we must sacrifice either **optimality**, **efficiency**, or **generality**
- Pseudo-polynomial time algorithms**: e.g., dynamic programming for the 0-1 Knapsack problem
- Exponential algorithms/Intelligent 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