UNIT 7
NP-COMPLETENESS

Iris Hui-Ru Jiang
Spring 2024

Department of Electrical Engineering
National Taiwan University

Outline

- Content:
  - Complexity classes
  - Reducibility and NP-completeness proofs
  - Coping with NP-complete problems
- Reading:
  - Chapter 34, 35.1~35.2

The Status of the P Versus NP Problem http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext

Decision Problems

- A computational problem can be viewed as a **function**, mapping an input to some output

- Decision problems: output = T/F (True/False, Yes/No)
  - Given a graph $G=(V, E)$, is it connected?
  - Given a graph $G=(V, E)$, u, $v \in V$, $k \in \mathbb{N}$, is there a path from u to v with $\leq k$ edges?
    - Decision version of shortest path problem
    - Instance: possible input; k: threshold/bound
  - Given a graph $G=(V, E)$, is there a Hamiltonian path/cycle?
    - A Hamiltonian path/cycle is a path/cycle which goes through every vertex exactly once

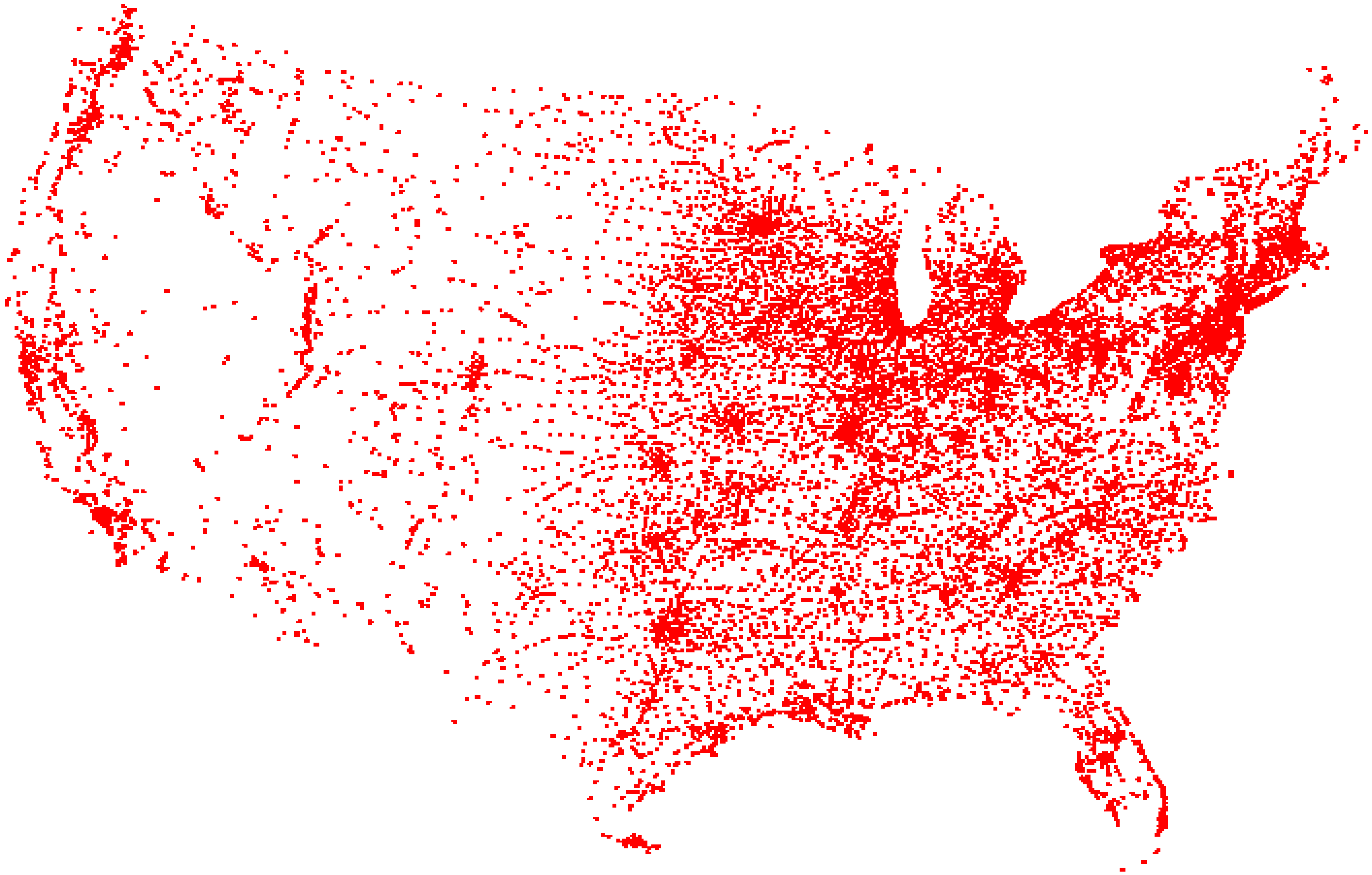
Decision vs. Optimization Problems

- **Decision** problems: those have yes/no answers
  - MST: Given a graph $G=(V, E)$ and a bound k, is there a spanning tree with a cost at most k?
  - TSP: Given a set of cities, distance between each pair of cities, and a bound b, is there a route that starts and ends at a given city, visits every city exactly once, and has total distance at most b?

- **Optimization** problems: those find a legal configuration such that its cost is minimum (or maximum)
  - MST: Given a graph $G=(V, E)$, find the cost of a minimum spanning tree of G.
  - TSP: Given a set of cities and the distance between each pair of cities, find the distance of a "minimum route" starts and ends at a given city and visits every city exactly once.

- Could apply **binary search** on the bound of a decision problem to obtain solutions to its optimization problem

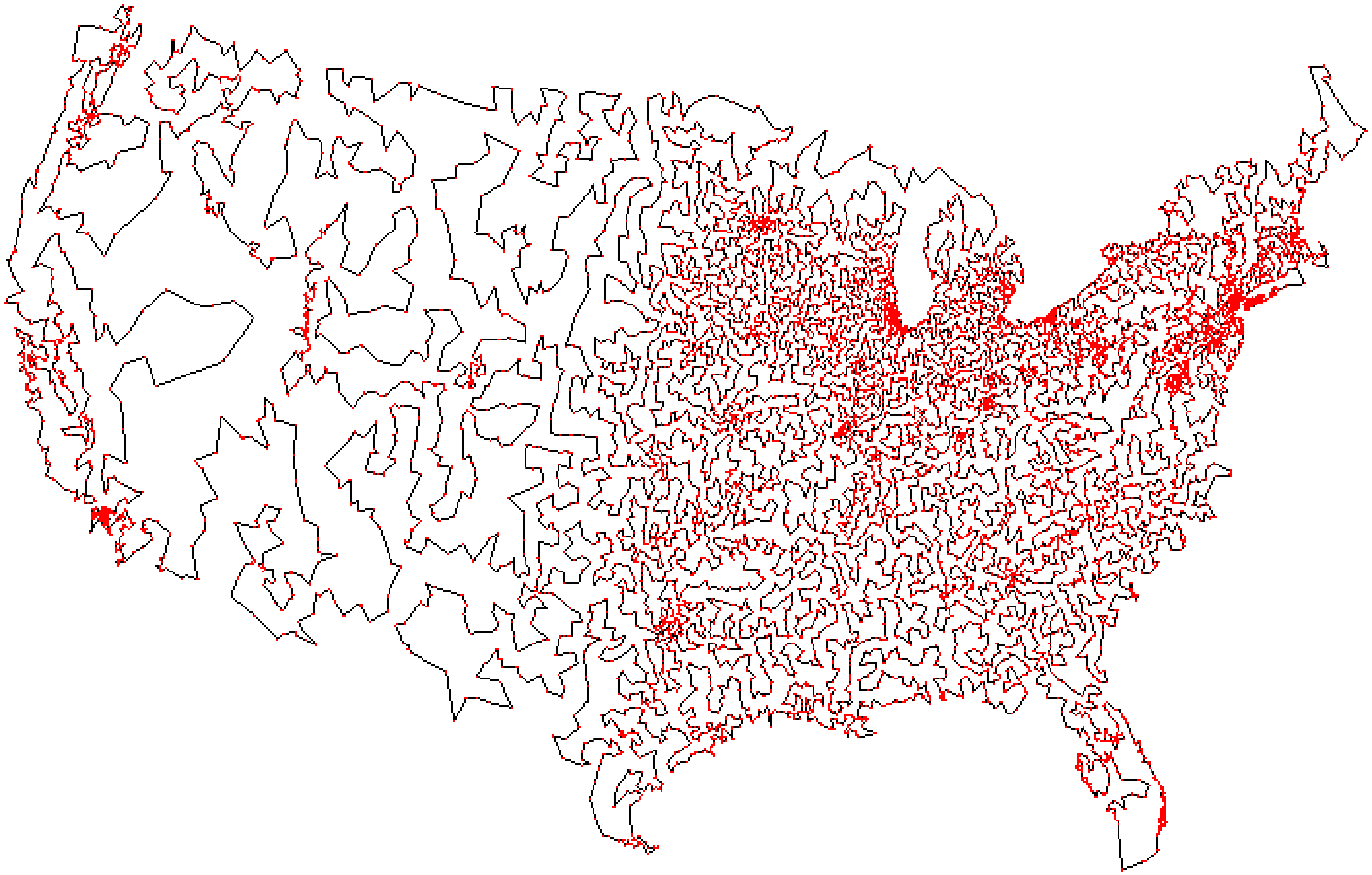
Traveling Salesman Problem (TSP) (1/2)



All 13,509 cities in US with a population of at least 500

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

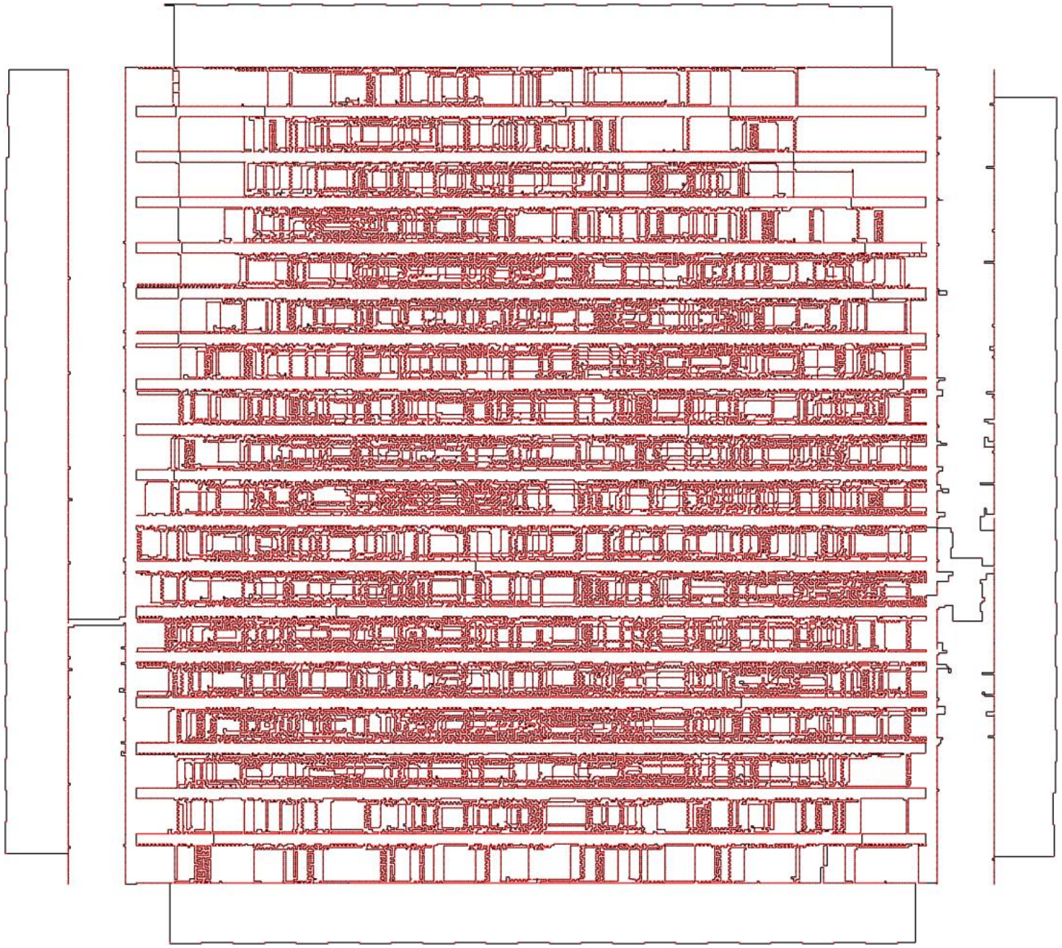
Traveling Salesman Problem (TSP) (2/2)



Optimal TSP tour

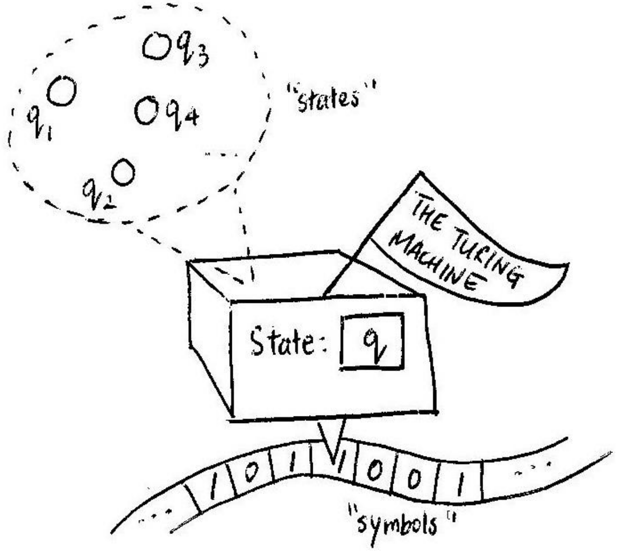
http://www.math.uwaterloo.ca/tsp/

TSP in VLSI

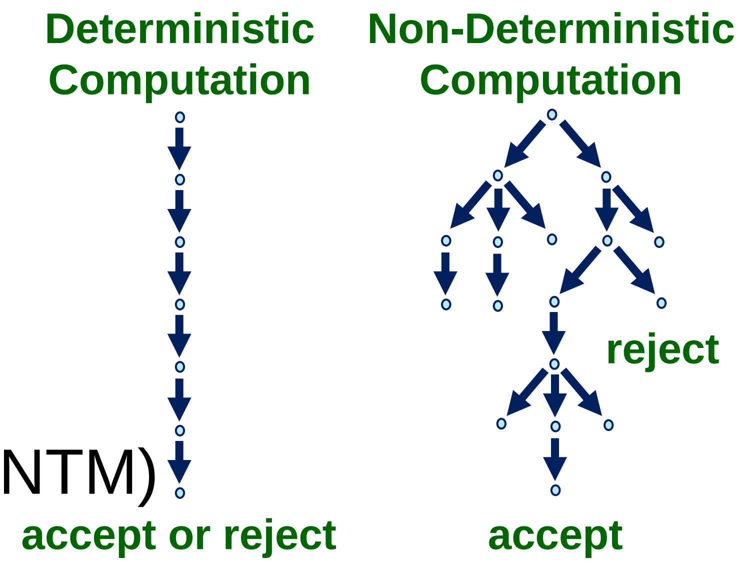


85,900 Locations in a VLSI Application
Solved in 2006

Turing Machine



- Deterministic Turing Machine (DTM)
  - Tape: (infinite memory)
    - Store inputs/outputs and results
    - ∞ long strip, divided into cells
    - One symbol each cell
  - Tape head:
    - Point to one cell at a time
    - Read a symbol from cell
    - Write a symbol to cell
    - Move one cell back/forward
  - Finite state machine (table)
- Nondeterministic Turing Machine (NTM)
  - Multiple branches



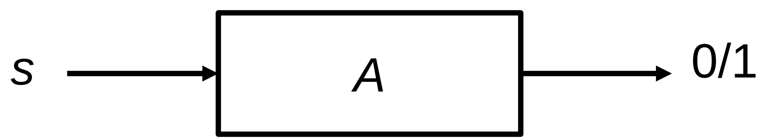
Think of a *Nondeterministic Turing Machine* as a computer that magically “guesses” a solution, then verifies it is correct. If a solution exists, computer always guesses it. *i.e.* a parallel computer that can freely spawn an infinite number of processes.

Complexity Class P

- A decision problem is in class P iff there is an algorithm *A* s.t.
- An instance *s* is true ⇔ *A*(*s*)=1
- *A* runs in polynomial time (of size of instance *s*)

⇔

- An instance *s* is false ⇔ *A*(*s*)=0



Always correct

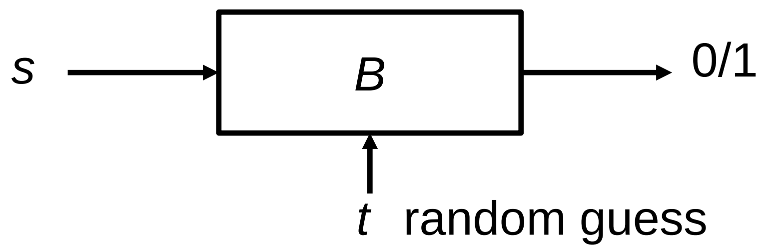
- P: polynomial-time solvable (of size of instance)

Complexity Class NP
Nondeterministic Polynomial Time

- A decision problem is in class NP iff there is an algorithm $B(s, t)$, which runs in polynomial time (of size of s), s.t.
- An instance s is true $\Leftrightarrow \exists t, B(s, t)=1$

$\Leftrightarrow$

- An instance s is false $\Leftrightarrow \forall t, B(s, t)=0$



If s is true, sometimes correct (depending on t)
If s is false, always correct

  – Considering infinite computing power (NTM), test each possible solution, e.g., $B(s, 0)$, $B(s, 1)$, $B(s, 00)$, $B(s, 01)$, …

- NP: polynomial-time verifiable (of size of instance)
  – Polynomial time solvable by NTM
  – Class NP is associated with decision problems

Example: Set Cover Problem

- Given
  - a set $U = \{a_1, a_2, \ldots, a_m\}$
  - n subsets $S_1, S_2, \ldots, S_n \subseteq U$
  - $k \in \mathbb{N}$

set cover

- Are there k subsets in $S_1, S_2, \ldots, S_n$ s.t. the union is U?
- e.g.,
  - U={1, 2, 3, 4, 5},S_1={1, 2, 3}, S_2={1, 3, 5}, S_3={3, 4}, S_4={2, 5}
  - k=2, false; k=3, true

	1	2	3	4	5
S_1	✖	✖	✖		
S_2	✖		✖		✖
S_3			✖	✖	
S_4		✖			✖

	1	2	3	4	5
S_1	✖	✖	✖		
S_3			✖	✖	
S_4		✖			✖

	1	2	3	4	5
S_2	✖		✖		✖
S_3			✖	✖	
S_4		✖			✖

SET COVER is NP

- **Theorem:** SET COVER $\in$ NP
- Pf: By definition
  - Idea: t: binary string, each bit indicates one subset
  - $B(s, t)$:
    - If t is not an n-bit binary number, $B(s, t)=0$
    - If the number of 1s in $t \neq k$, $B(s, t)=0$
    - Otherwise, pick S_i iff t[i]=1 (will exactly pick k subsets)
  - e.g., $B(s,1011)=1$, $B(s,0111)=1$, for $k=3$

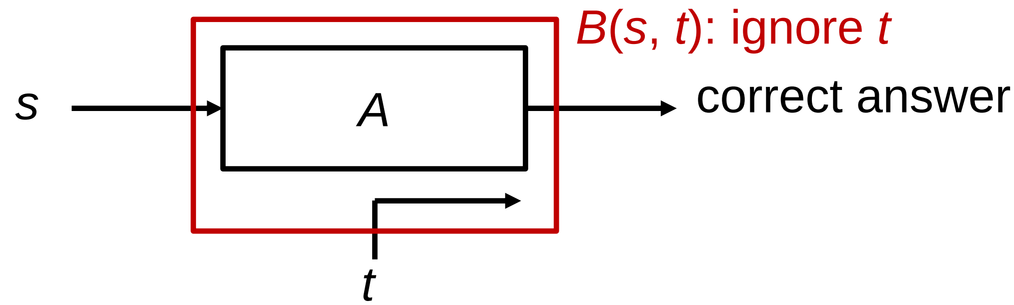
	1	2	3	4	5
S_1	X	X	X		
S_2	X		X		X
S_3			X	X	
S_4		X			X

	1	2	3	4	5
S_1	X	X	X		
S_3			X	X	
S_4		X			X

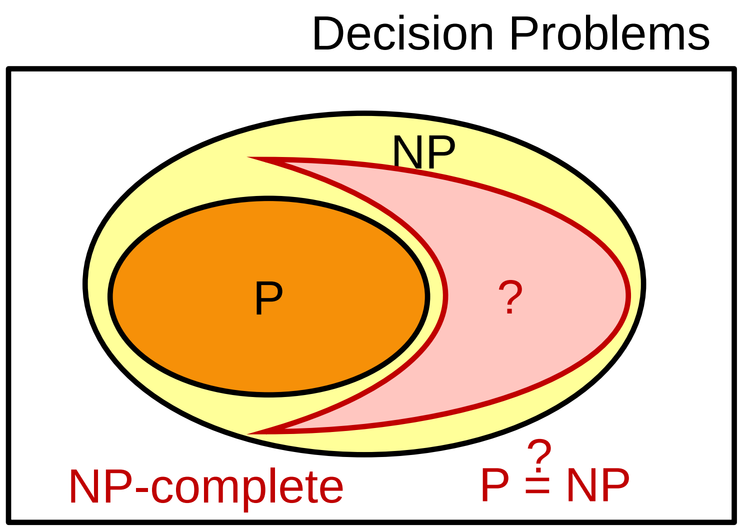
	1	2	3	4	5
S_2	X		X		X
S_3			X	X	
S_4		X			X

Relation between P and NP

- **Theorem:** If problem $X \in$ P, then $X \in$ NP
- Pf:
  - $X \in P$, $\exists$ A, $A(s)$ is always correct



  - Take $B(s, t)=A(s)$
  - $B(s, t)$ satisfies the requirement of NP
  - $X \in$ NP



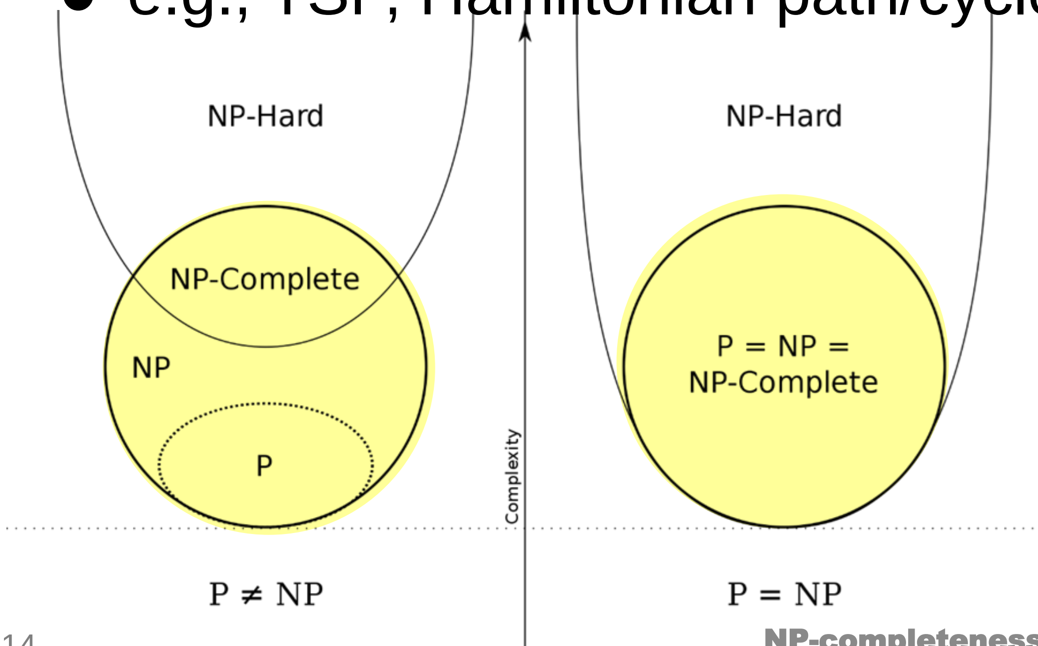
NP-Completeness (NPC)

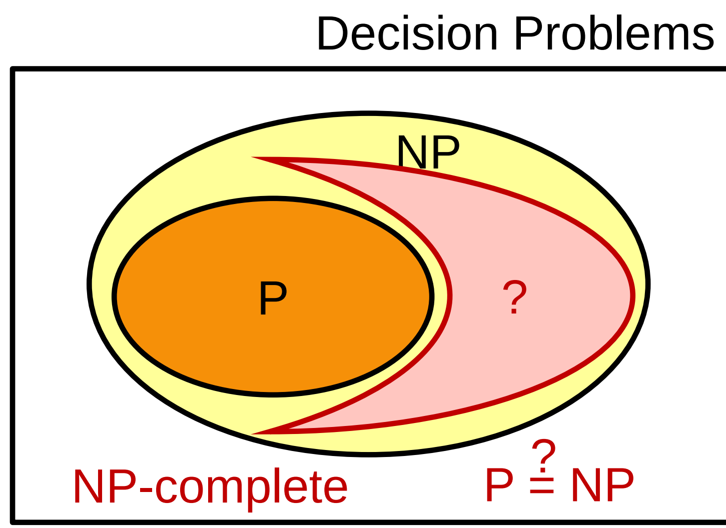
Informal Definition

- A decision problem is in class NP-complete iff

1. It is NP, and
2. If any NP-complete problem is in P then P=NP
   - NP=P, i.e., every problem in NP has polynomial time algorithms

- NPC: hardest problems in NP
- e.g., TSP, Hamiltonian path/cycle, set cover, Tetris, Sudoku

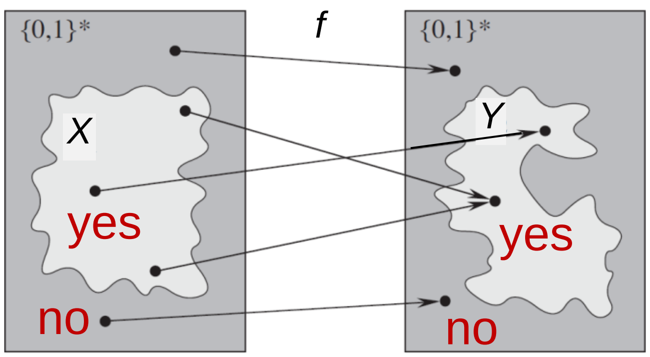


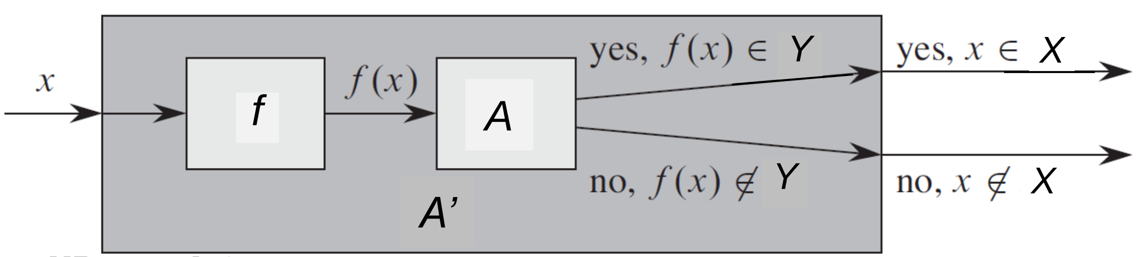


Polynomial-Time Reduction

Notation: $X \leq_P Y$

- **Motivation:** Let *X* and *Y* be two decision problems. If algorithm *A* can solve *Y*, can we use *A* to solve *X*?
- Polynomial Time Reduction *f* from *X* to *Y*: $X \leq_P Y$
  1. For any instance *x* of *X*, *f*(*x*)=*y* is an instance of *Y*
  2. *x* is true iff *y* is true
  3. Function (mapping) *f* can be computed in polynomial time of size of *x*
- Remarks:
  - *Y* is harder than or equally hard as *X*
  - Multiple instances of *X* may share the same instance of *Y*
  - The algorithm for *Y* is viewed as a black box. We pay for polynomial time to write down instances sent to this black box.





NP-completeness

Polynomial-Time Reduction

Purpose: Classify problems according to relative difficulty

1. Design algorithms: If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can be solved in polynomial time
   – Bipartite matching $\leq_P$ Network flow
   – System of difference constraint $\leq_P$ SSSP (Bellman-Ford)

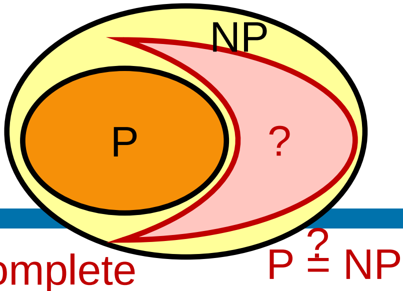
2. Establish intractability: If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time
   – SAT $\leq_P$ Set cover
   – Hamiltonian cycle $\leq_P$ Travelling salesman

3. Establish equivalence: If $X \leq_P Y$ and $Y \leq_P X$, $X \equiv_P Y$
   – Up to cost of reduction

- $\leq_P$ is transitive: $X \leq_P Y$ and $Y \leq_P Z \Rightarrow X \leq_P Z$

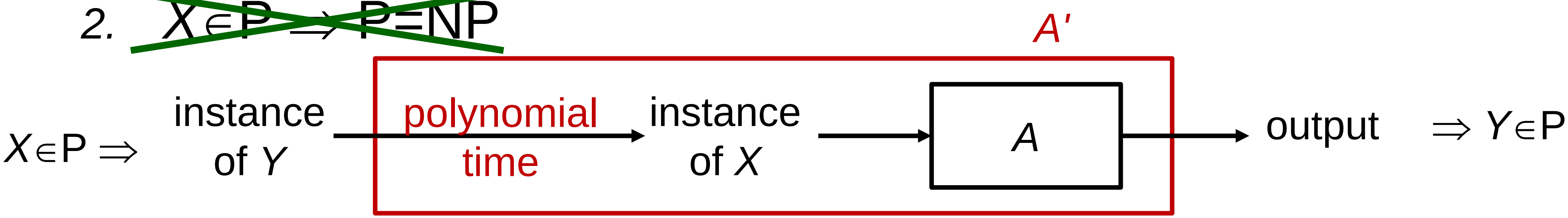
NP-Completeness (NPC)

Definition and Theorem



- **Definition:** A decision problem X is NP-complete iff

1. $X \in NP$
2. $\cancel{X \in P \Rightarrow P = NP}$



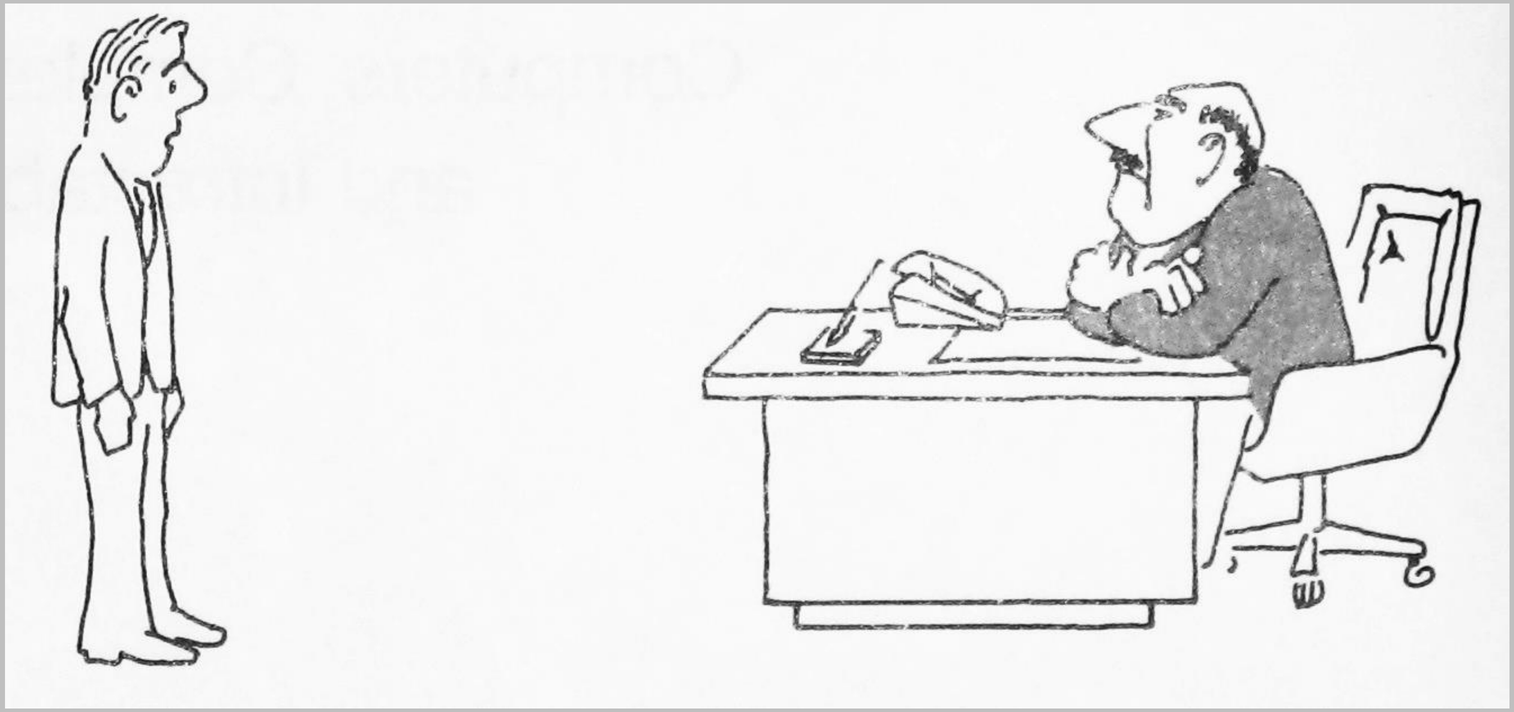
2. $\forall Y \in NP,\ Y \leq_P X$

- **Theorem:** A decision problem X is NP-complete iff

1. $X \in NP$
2. $\exists Y \in NPC,\ Y \leq_P X$
   - NP hard: not required to be NP, e.g., halting problem

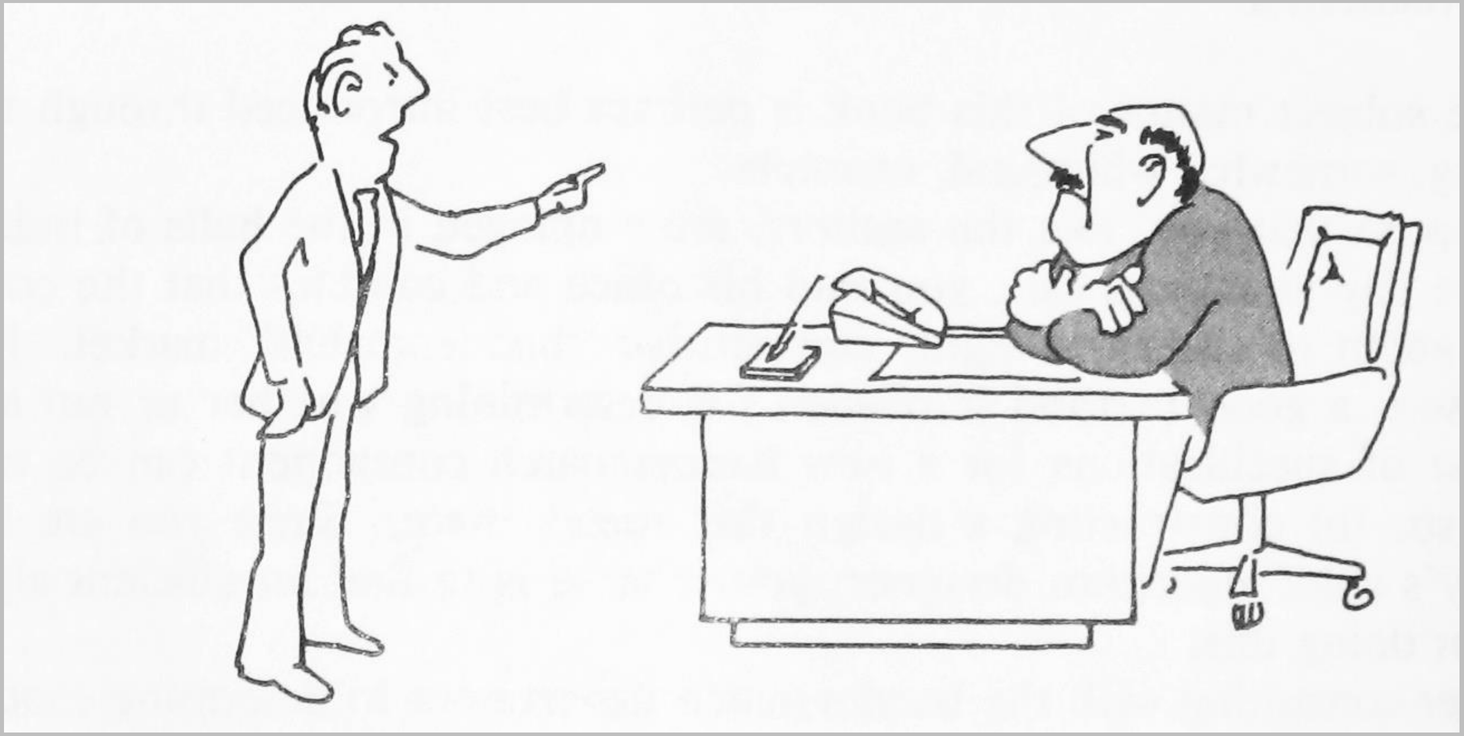
- Pf:
  - All NP $\leq_P Y \leq_P X$

What You'd Rather Not Say



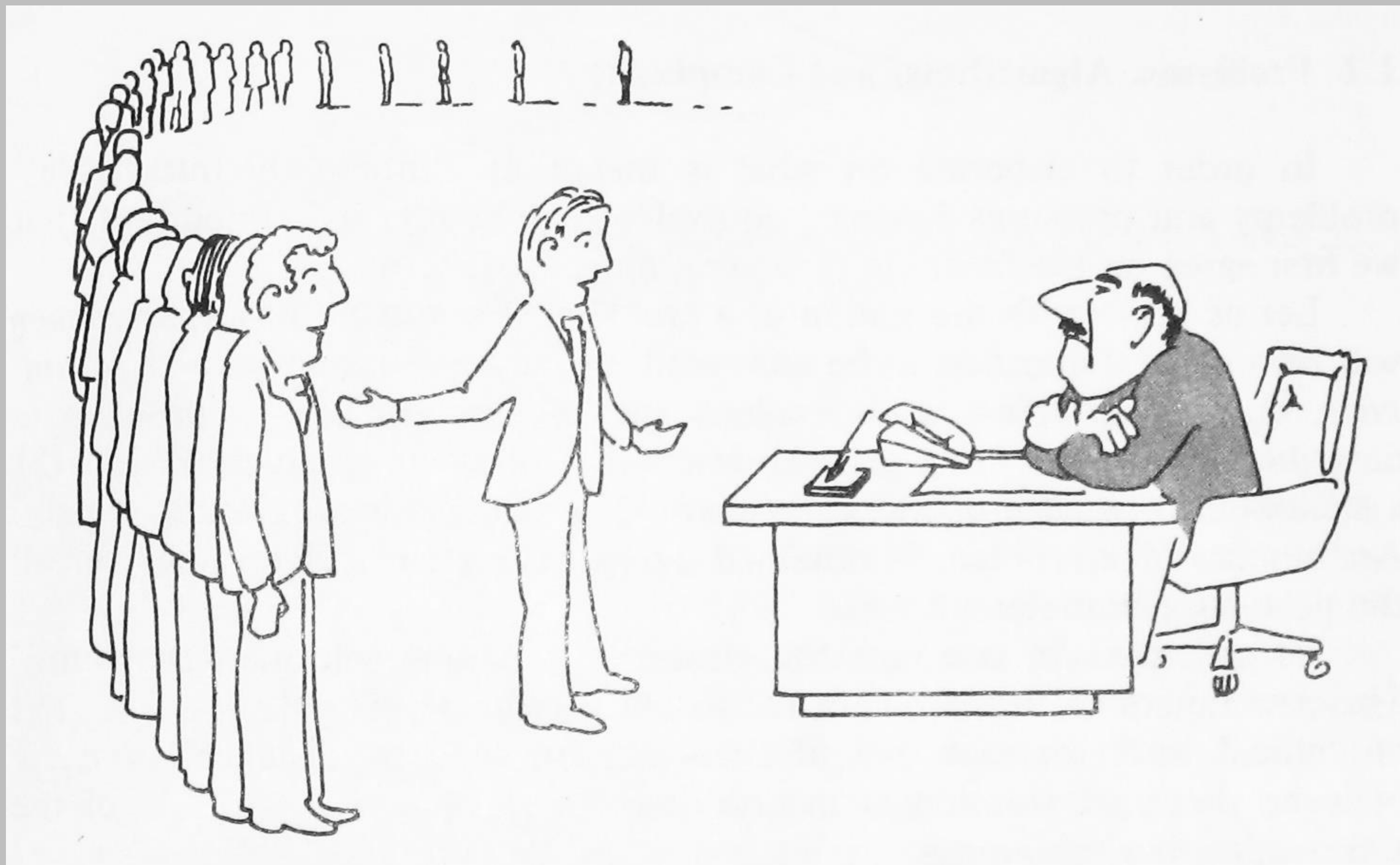
"I can't find an efficient algorithm. I guess I'm just too dumb."

You Cannot Say This, Either



"I can't find an efficient algorithm, because no such algorithm is possible!"

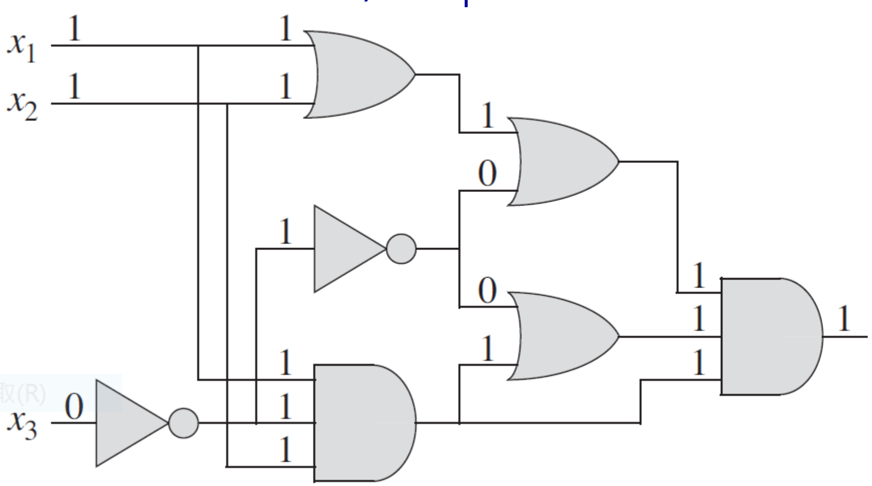
What You Can Say



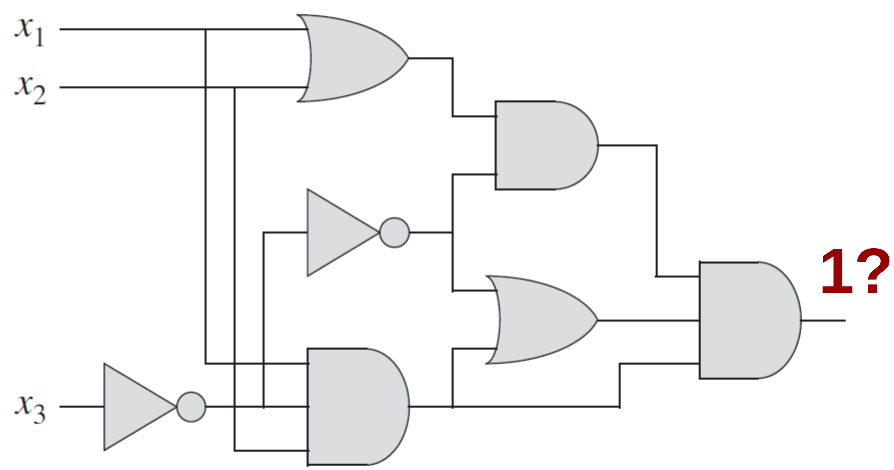
“I can’t find an efficient algorithm, but neither can all these famous people.”

The Circuit-Satisfiability Problem (Circuit-SAT)

- The First Proved NPC: Circuit Satisfiability
- **The Circuit-Satisfiability Problem (Circuit-SAT):**
  - **Instance:** A combinational circuit *C* composed of AND, OR, and NOT gates.
  - **Question:** Is there an assignment of Boolean values to the inputs that makes the output of *C* equal to 1?
- A circuit is satisfiable if there exists a set of Boolean input values that makes the output of the circuit to be 1.
- **Circuit-SAT is NP-complete.** (Cook, ACM STOC'71)
  - Circuit-SAT $\in$ NP
  - $\forall L' \in$ NP, $L' \leq_P$ Circuit-SAT



satisfiable



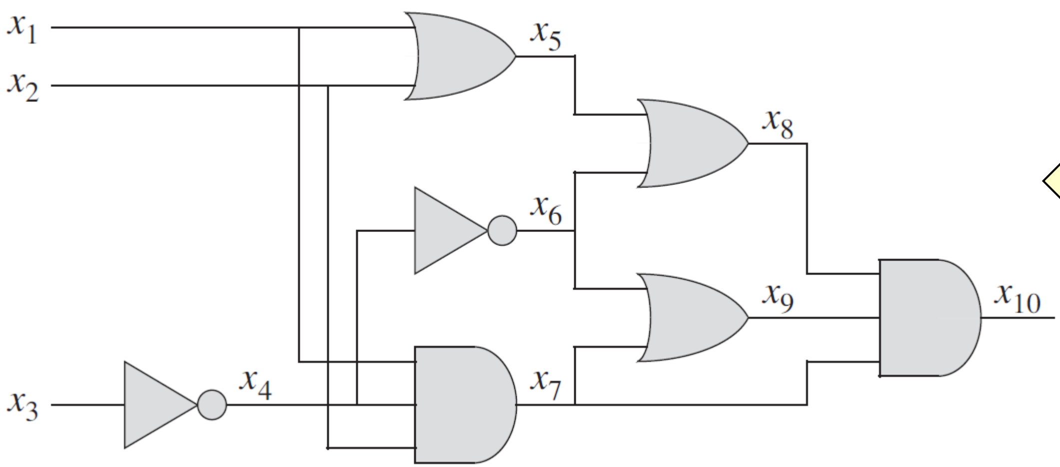
unsatisfiable

The Satisfiability Problem (SAT)

- The Satisfiability Problem (SAT):
  - **Instance:** A Boolean formula ϕ
  - n Boolean variables: $x_1, x_2, x_3, \ldots x_n$
  - m Boolean connectives: AND, OR, NOT, $\leftrightarrow$, $\rightarrow$, parentheses
  - **Question:** Is there an assignment of truth values to the variables that makes ϕ true?
- **Truth assignment:** set of values for the variables of ϕ
- **Satisfying assignment:** a truth assignment that makes ϕ evaluate to 1
- Exp: $\phi = ((x_1 \rightarrow x_2) \vee \neg ((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - Truth assignment: $<x_1, x_2, x_3, x_4> = <0, 0, 1, 1>$, $<0, 1, 0, 1>$, etc.
  - Satisfying assignment: $<x_1, x_2, x_3, x_4> = <0, 0, 1, 1>$, etc.
- **Satisfiable formula:** a formula with a satisfying assignment.
  - ϕ is a satisfiable formula.

SAT is NP-Complete

1. **(Formula) SAT $\in$ NP.**
2. **(Formula) SAT is NP-hard:** Prove that Circuit-SAT $\leq_P$ SAT.
   - For each wire x_i in circuit C, the formula ϕ has a variable x_i.
   - The operation of a gate is expressed as a formula with associated variables, e.g., $\boldsymbol{x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)}$.
   - ϕ = AND of the circuit-output variable with the **conjunction** ($\wedge$) of clauses describing the operation of each gate, e.g.,
   - Circuit C is satisfiable $\Leftrightarrow$ formula ϕ is satisfiable. (Why?)
   - Given a circuit C, it takes polynomial time to construct ϕ.



$$\phi = x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \wedge \underbrace{(x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))}_{\text{clause}}$$

Reduction from Circuit-SAT to formula SAT

SAT $\leq_P$ SET COVER

Conjunctive Normal Form

- Given an SAT instance of CNF form $c_1 \wedge c_2 \ldots \wedge \ldots c_m$ using variables $x_1, x_2, \ldots x_n$
- Construct the corresponding SET COVER instance

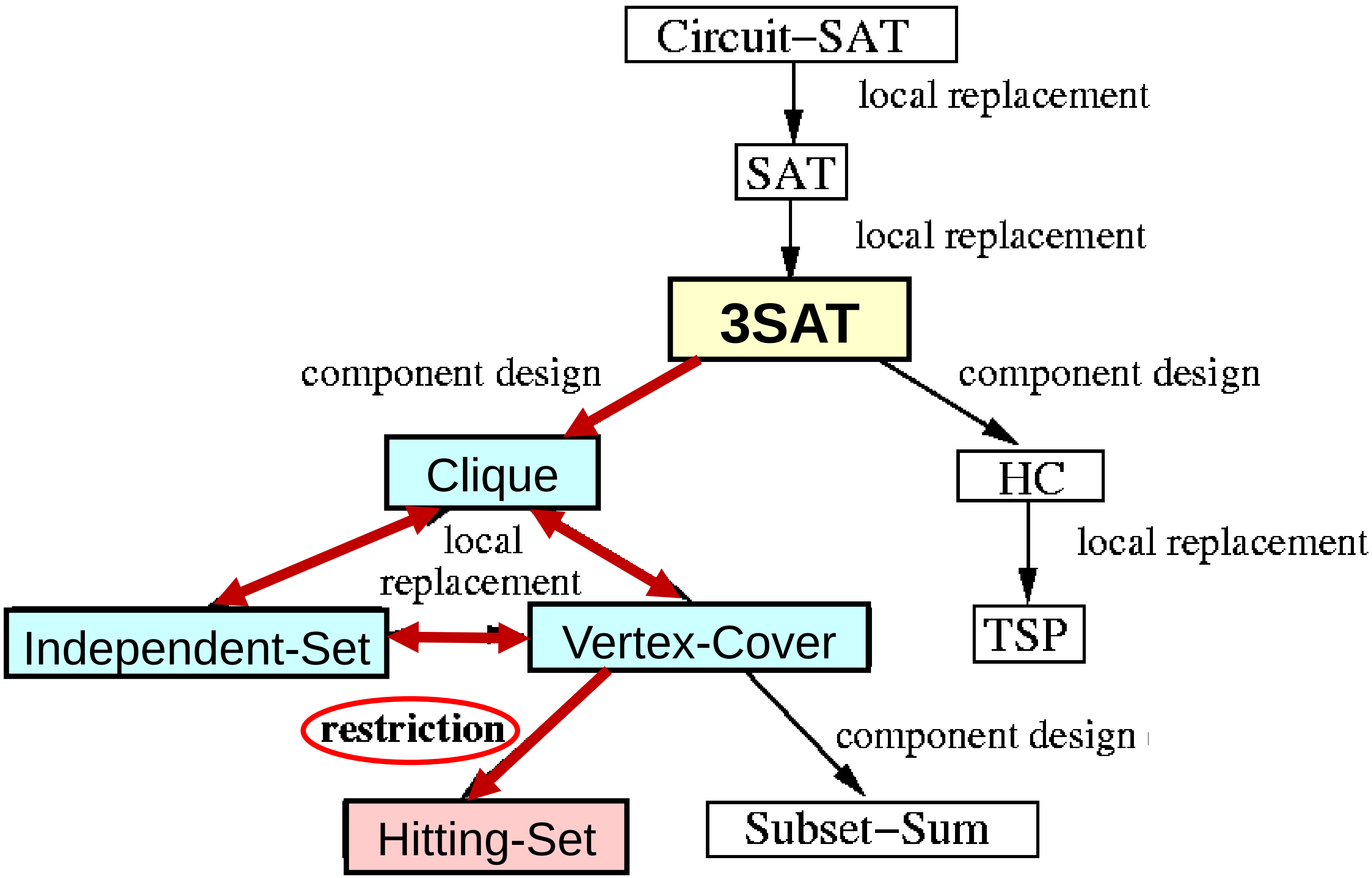
  $U=\{\underbrace{d_1, d_2, \ldots d_m}_{\text{clause}}, \underbrace{y_1, y_2, \ldots y_n}_{\text{variables}}\}$,

  $k=n$,

  subsets $S_{1T}, S_{1F}, S_{2T}, S_{2F}, \ldots, S_{nT}, S_{nF}$
- e.g., $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2)$
  - $U=\{d_1, d_2, d_3, y_1, y_2, y_3\}$
  - $\boxed{S_{1T} = \{d_1, d_2, y_1\}}$: set x_1 as 1, which clauses are true?
  - $S_{1F} = \{d_3, y_1\}$: set x_1 as 0, which clauses are true?
  - $S_{2T} = \{d_1, y_2\}$
  - $\boxed{S_{2F} = \{d_2, d_3, y_2\}}$
  - $\boxed{S_{3T} = \{d_2, y_3\}}$
  - $S_{3F} = \{d_1, y_3\}$

x_1=T; x_2=F; x_3=T

Structure of NP-Completeness Proofs



3SAT is NP-Complete

- 3SAT: Satisfiability of Boolean formula in 3-conjunctive normal form (3-CNF)
  - Each clause has exactly 3 distinct literals, e.g.,

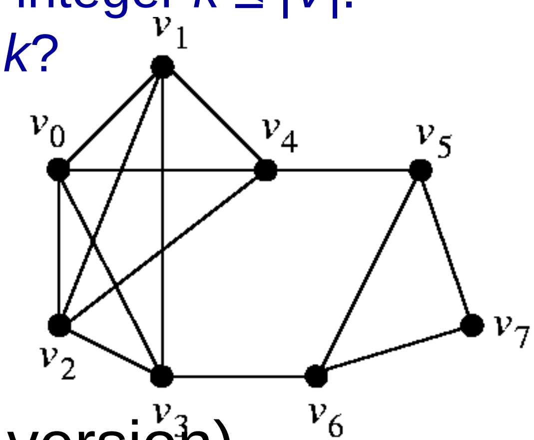
$$\phi\ (x_1 \underset{\text{OR}}{\vee} x_2 \vee \neg x_3) \underset{\text{AND}}{\wedge} (x_1 \vee \underset{\text{NOT}}{\neg} x_2 \vee x_3) \wedge \underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_4)}_{\text{clause}} \wedge \ldots$$

- **3SAT $\in$ NP** (will omit this part for other proofs)
- **3SAT is NP-hard:** SAT $\leq_P$ 3SAT (see appendix)

1. Construct a binary "parse" tree for input formula ϕ and introduce a variable y_i for the output of each internal node.
2. Rewrite ϕ as the AND of the root variable and a conjunction of clauses describing the operation of each node.
3. Convert each clause ϕ'_i into CNF.

Clique is NP-Complete

- A **clique** in $G = (V, E)$ is a complete subgraph of G.
- **The Clique Problem (Clique)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - **Question:** is there a clique $V' \subseteq V$ of size $\geq k$?
- Example:
  - Cliques: $\{v_0, v_1, v_2, v_4\}$ $\{v_3, v_6\}$ $\{v_6, v_5, v_7\}$ …
  - Is there clique of size=4? Yes
    - $\{v_0, v_1, v_2, v_4\}$



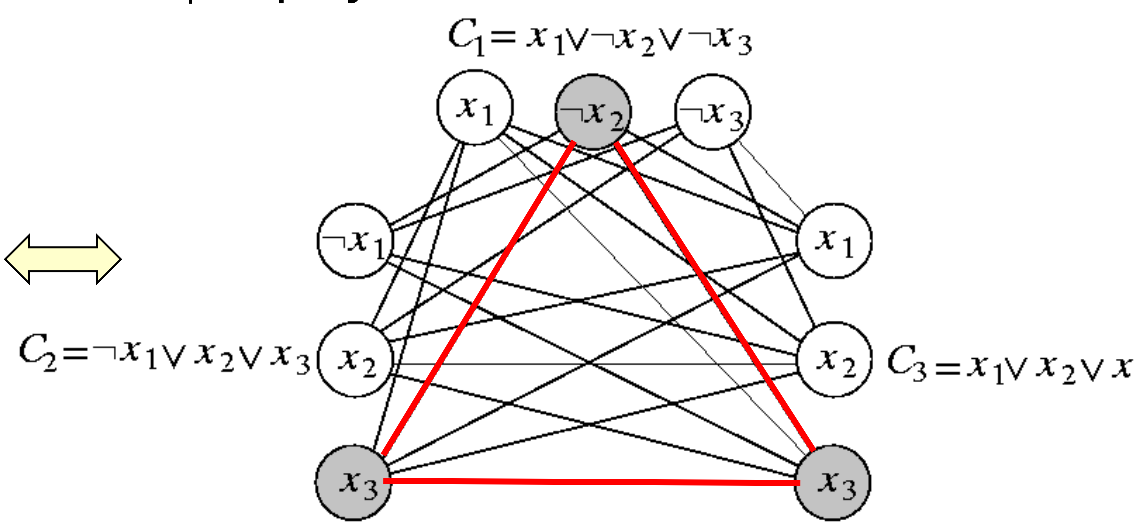
- Maximum clique problem (optimization version)
  - Given an undirected graph $G = (V,E)$ find maximum clique

- **Clique $\in$ NP.**
- **Clique is NP-hard:** 3SAT $\leq_P$ Clique.
  - **Key:** Construct a graph G such that ϕ is satisfiable $\Leftrightarrow$ G has a clique **of size k**.

3SAT $\leq_P$ Clique

- Let $\phi = \mathbf{C_1} \wedge \mathbf{C_2} \wedge \ldots \wedge \mathbf{C_k}$ be a Boolean formula in 3-CNF with k clauses. Each C_r has exactly 3 distinct literals l_1^r, l_2^r, l_3^r.
- For each $\mathbf{C_r = (l_1^r \vee l_2^r \vee l_3^r)}$ in ϕ, introduce a triple of vertices v^r_1, v^r_2, v^r_3 in V.
- Build an edge (compatible assignment) 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 (No edge between x_3 and $\neg x_3$ (inconsistent))
- G can be constructed from ϕ in polynomial time.

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

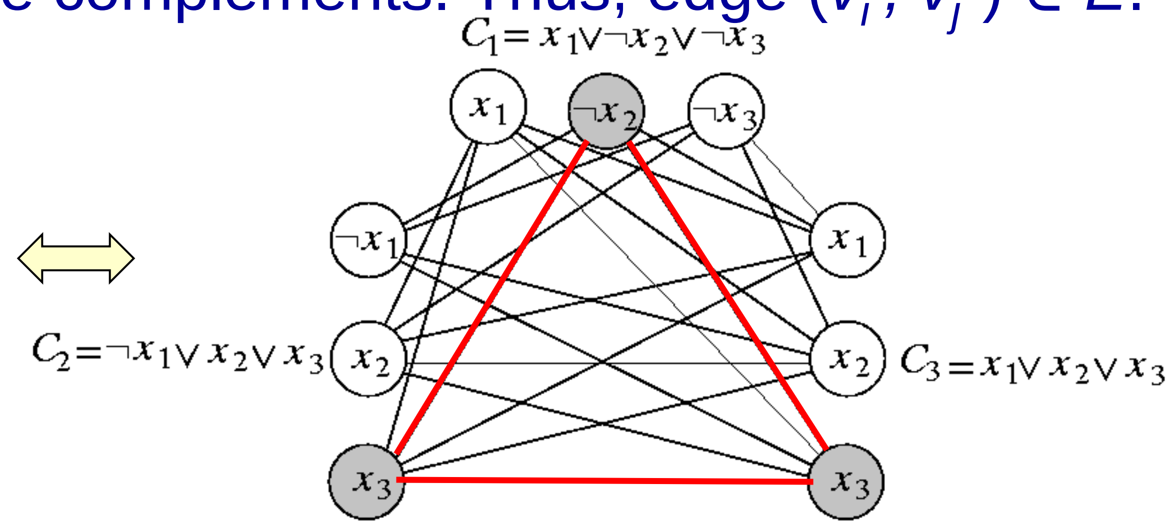


Satisfying assignment: $\langle x_1, x_2, x_3 \rangle = \langle x, 0, 1 \rangle$

ϕ Is Satisfiable $\Leftrightarrow$ *G* Has a Clique of Size *k*

- $\phi = C_1 \wedge \ldots \wedge C_k$ is satisfiable $\Rightarrow$ *G* has a clique of **size *k***.
  - 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.
  - Picking a "true" literal from each C_r forms a set of V' of k vertices.
  - For any two vertices $v_i^r, v_j^s \in V'$, $r \neq s$, $l_i^r = l_j^s = 1$ and thus l_i^r, l_j^s cannot be complements. Thus, edge $(v_i^r, v_j^s) \in E$.

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

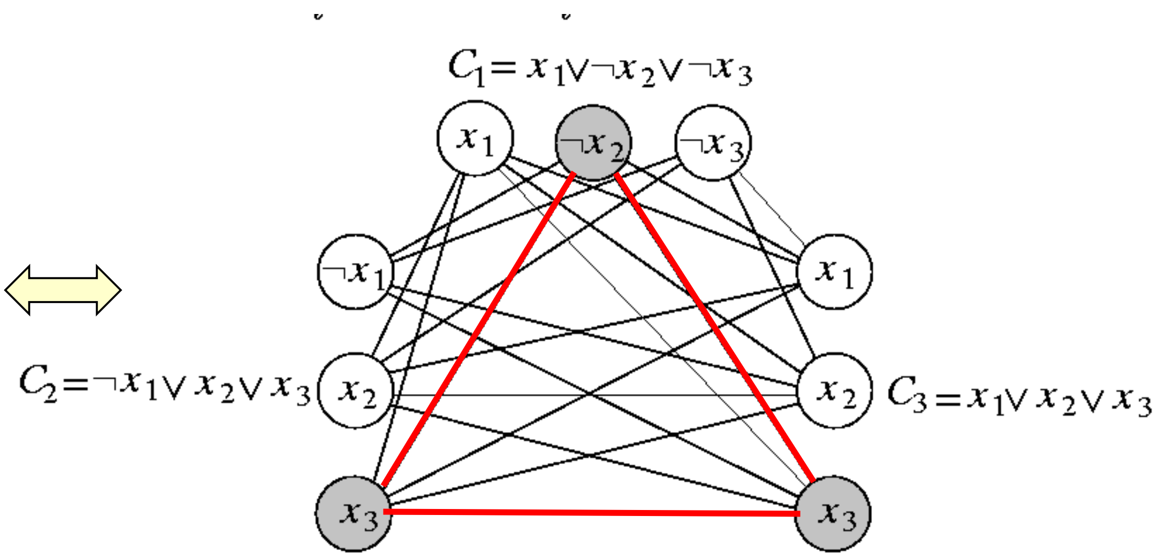


Satisfying assignment: $\langle x_1, x_2, x_3 \rangle = \langle x, 0, 1 \rangle$

ϕ Is Satisfiable $\Leftrightarrow$ *G* Has a Clique **of Size *k***

- *G* has a clique of **size *k*** $\Rightarrow$ ϕ is satisfiable
  - *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.
  - Assign 1 to each l_i^r such that $v_i^r \in V'$ $\Rightarrow$ each C_r is satisfied, and so is ϕ.

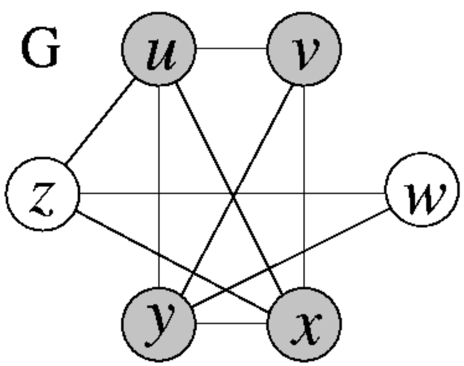
$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

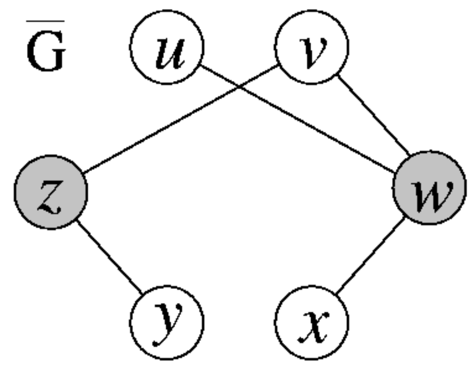


Satisfying assignment: $<x_1, x_2, x_3> = <x, 0, 1>$

Vertex-Cover is NP-Complete

- 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 $v \in V'$. (Vertices cover edges)
- **The Vertex-Cover Problem (Vertex-Cover)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \le |V|$.
  - **Question:** is there a subset $V' \subseteq V$ of size $\le k$ such that each edge in E has at least one vertex (endpoint) in V'?
- **Vertex-Cover $\in$ NP.**
- **Vertex-Cover is NP-hard:** Clique $\le_P$ Vertex-Cover.
  - **Key: complement** of G: $\bar{G} = (V, \bar{E})$, $\bar{E} = \{(w, v): (w, v) \notin E\}$.

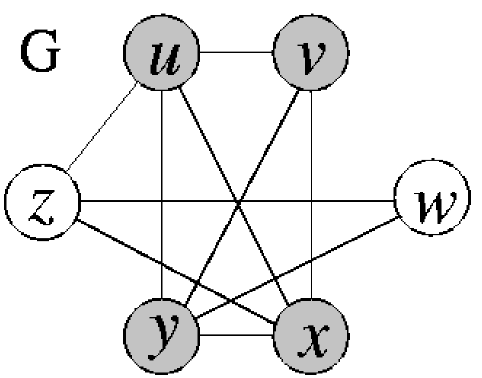




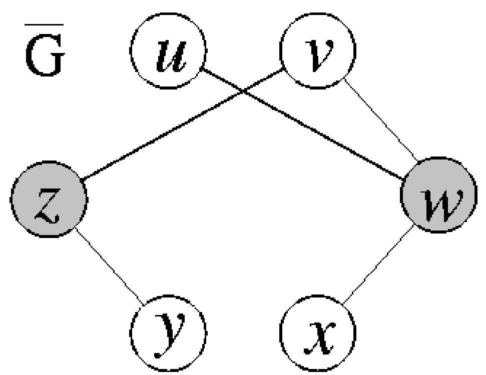
Clique $V' = \{u, v, x, y\}$ Vertex cover $V - V' = \{w, z\}$

Vertex-Cover is NP-Complete (cont'd)

- G Has a Clique of Size $k \Rightarrow \bar{G}$ Has a Vertex Cover of size $|V| - k$.
  - Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$.
  - Let (w, v) be any edge in $\bar{E} \Rightarrow (w, v) \notin E \Rightarrow$ at most one of w or v in $V' \Rightarrow$ at least one of w or v does not belong to V'
  - So, $w \in V - V'$ or $v \in V - V' \Rightarrow$ edge (w, v) is covered by $V - V'$.
  - Thus, $V - V'$ forms a vertex cover of $\bar{G}$, and $|V - V'| = |V| - k$.



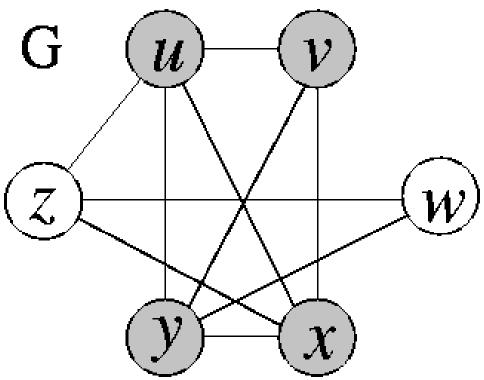
Clique $V' = \{u, v, x, y\}$



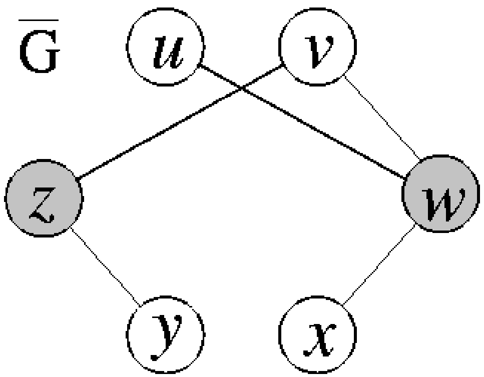
Vertex cover $V - V' = \{w, z\}$

Vertex-Cover is NP-Complete (cont'd)

- $\bar{G}$ Has a Vertex Cover of size $|V|-k$ $\Rightarrow$ G Has a Clique of Size k.
  - Suppose that $\bar{G}$ has a vertex cover $V' \subseteq V$ with $|V'| = |V| - k$.
  - $\forall\ a, b \in V$, if $(a, b) \in \bar{E}$, then $a \in V'$ or $b \in V'$ or both.
  - So, $\forall\ a, b \in V$, if $a \notin V'$ and $b \notin V'$, $(a, b) \in E \Rightarrow V - V'$ is a clique, and $|V| - |V'| = k$.



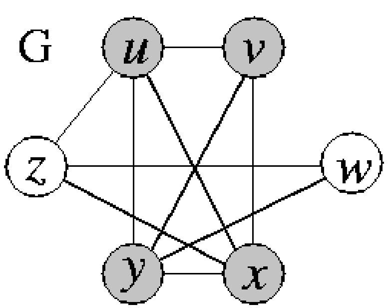
Clique $V - V' = \{u, v, x, y\}$



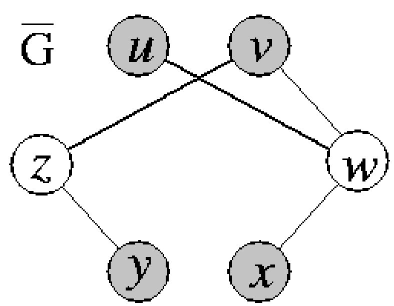
Vertex cover $V' = \{w, z\}$

Clique, Independent-Set, Vertex-Cover

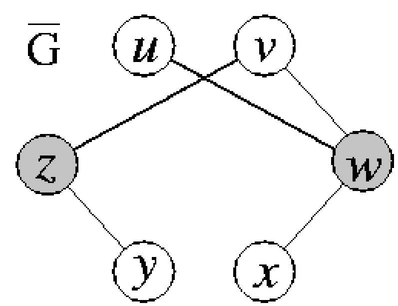
- An **independent set** of $G = (V, E)$ is a subset $V' \subseteq V$ such that G has no edge between any pair of vertices in V'
- **The Independent-Set Problem (Independent-Set)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \leq |V|$
  - **Question:** is there an independent set of size $\geq k$?
- **Theorem:** 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 $\bar{G}$
  3. $V-V'$ is a vertex cover of $\bar{G}$
- **Corollary:** Independent-Set is NP-complete



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



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



Vertex cover $V - V' = \{w, z\}$

Restriction: Hitting-Set is NP-Complete

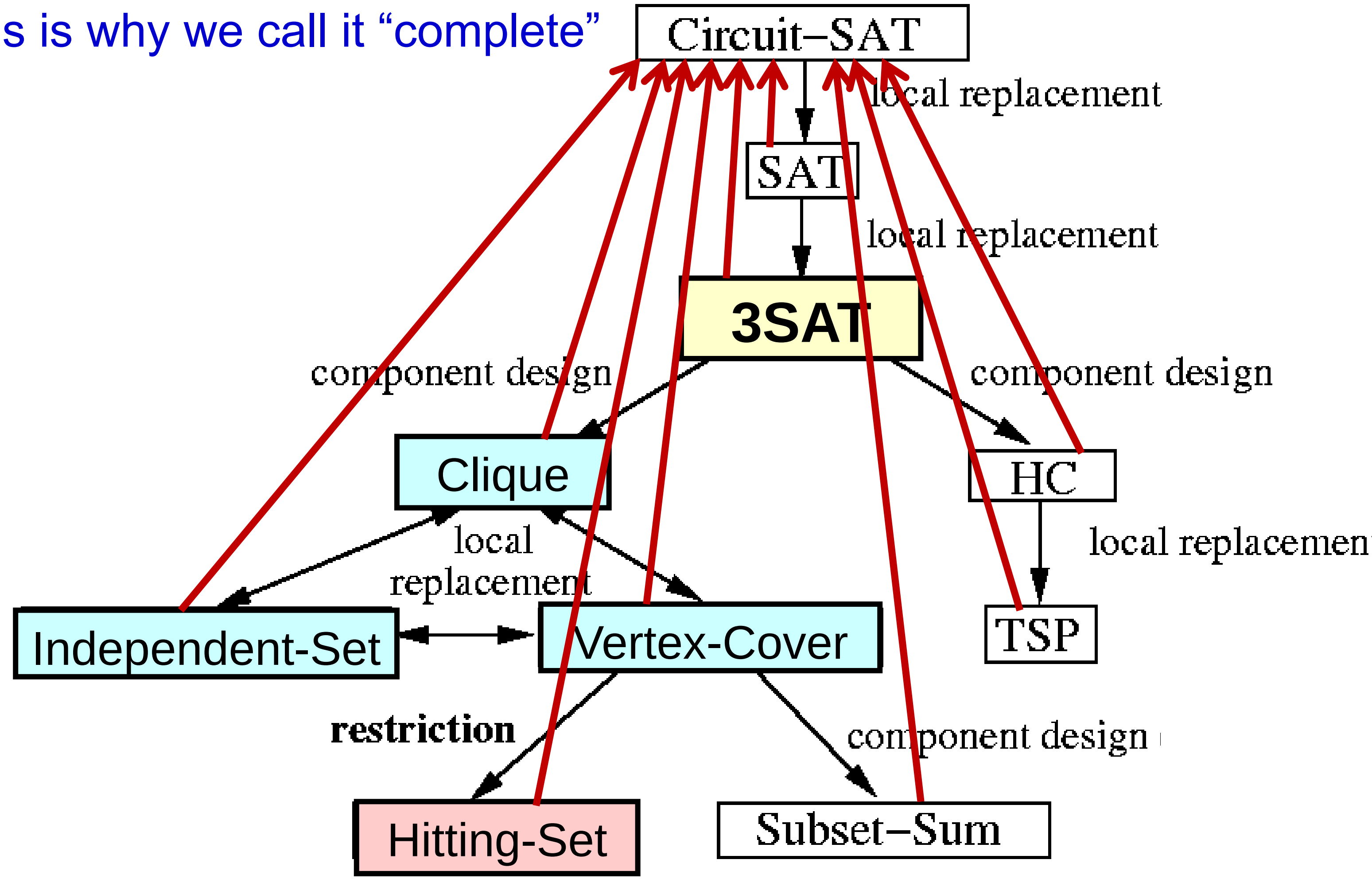
- A **hitting set** for a collection *C* of subsets of a set *S* is a subset $S' \subseteq S$ such that S' contains at least one element from each subset in *C*.
  - $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $C = \{\{1\}, \{3, 5\}, \{4, 7, 8\}, \{5, 6\}\}$
    S' can be $\{1, 4, 5\}$, $\{1, 3, 4, 6\}$, etc.
- **The Hitting-Set Problem (Hitting-Set)**
  - **Instance:** Collection *C* of subsets of a set *S*, positive integer *k*.
  - **Question:** Does *S* contain a hitting set for *C* of size $\leq k$?
- **Hitting-Set is NP-Complete**.
  - Restrict to Vertex-Cover by allowing only instances having $|c| = 2$ for all $c \in C$.
  - Each set in $C \leftrightarrow$ edge; element in $S' \leftrightarrow$ vertex cover.
- **Proof by restriction** is the simplest, and perhaps the most frequently used technique.
  - Other examples: Bounded Degree Spanning Tree, Directed HC, Weighted HC, Longest Simple Cycle, etc.

Summary

- The class P: class of problems that can be solved in polynomial time in the size of input.
- The class NP (Nondeterministic Polynomial): class of problems that can be verified in polynomial time in the size of input.
  - P=NP?
- The class NP-complete (NPC): A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.
- Theorem: Suppose Y is NPC, then Y is solvable in polynomial time iff P = NP.
  - Any NPC problem can be solved in polynomial time $\Rightarrow$ All problems in NP can be solved in polynomial time.

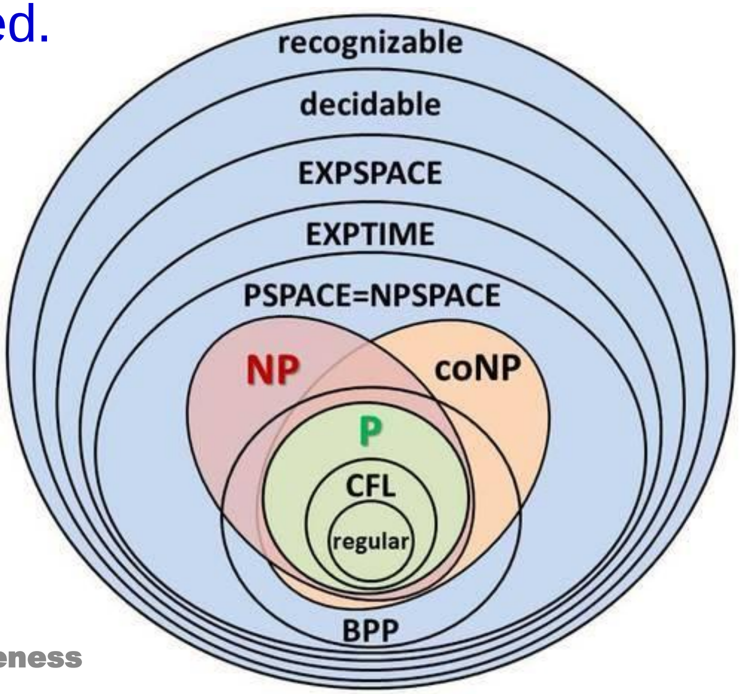
NPC Problems Are Equally Hard!

- All NP problems can be reduced to Circuit-SAT
  - This is why we call it "complete"



Other Complexity Classes

- Many other complexity classes...
  - co-NP: Problems whose complement is NP
  - PSPACE: Problems that can be solved using a polynomial memory space, regardless of computation time
  - EXPTIME: Problems that can be solved in exponential time
  - Undecidable: there is no algorithm that solves them, no matter how much time or space is allowed.
    - Halting problem



Coping with NP-Complete/-Hard Problems

- **Approximation algorithms:**
  - Guarantee to be a fixed percentage away from the optimum.
- **Pseudo-polynomial time algorithms:**
  - E.g., dynamic programming for the 0-1 Knapsack problem.
- **Probabilistic algorithms:**
  - Assume some probabilistic distribution of the instances.
- **Randomized algorithms:**
  - Make use of a randomizer (random # generator) for operation.
- **Restriction:** Work on some special cases of the original problem.
  - E.g., the maximum independent set problem in circle graphs.
- **Exponential algorithms/Branch and Bound/Exhaustive search:**
  - Feasible only when the problem size is small.
- **Local search:**
  - Simulated annealing (hill climbing), genetic algorithms, etc.
- **Heuristics:** No formal guarantee of performance.

Approximation Algorithms

- **Approximation algorithm:** An algorithm that returns **near-optimal** solutions.
- **Ratio (Performance) bound $\rho(n)$:** For any input size n, the cost C of the solution produced by an approximation algorithm $\leq \rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

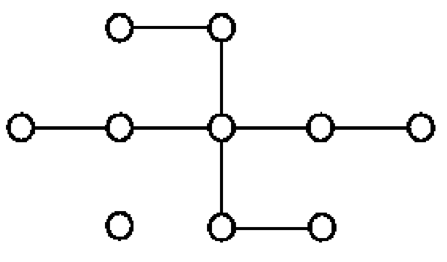
  – $\rho(n) \geq 1$.
  – An optimal algorithm has ratio bound 1.
- **Relative error bound $\varepsilon(n)$:**

$$\frac{|C - C^*|}{C^*} \leq \epsilon(n).$$

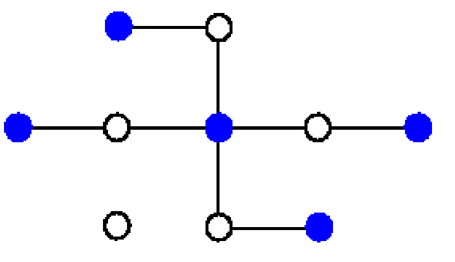
  – $\varepsilon(n) \leq \rho(n) - 1$.

Greedy Vertex Cover Algorithm Revisited

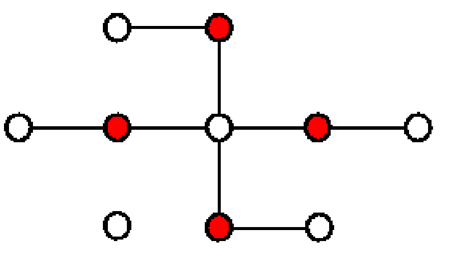
- **Greedy heuristic:** cover as many edges as possible (vertex with the maximum degree) at each stage and then delete the covered edges.
- **The greedy heuristic cannot always find an optimal solution!**
  - The vertex-cover problem is NP-complete.
- **The greedy heuristic cannot guarantee a constant performance bound.**



A graph instance



A vertex cover of size 5 by the greedy algorithm



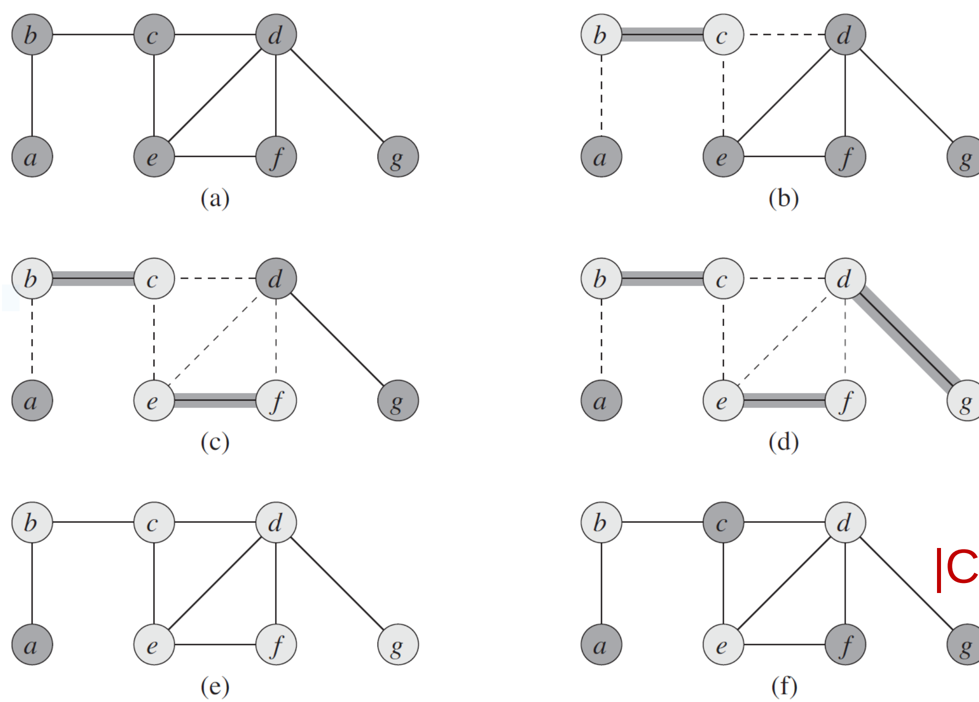
A vertex cover of size 4 **Optimal solution!!**

The Vertex-Cover Problem

```
Approx-Vertex-Cover(G)
C = ∅
E' = E[G]
while E' ≠ ∅
      let (u, v) be an arbitrary edge of E'
      C = C ∪ {u, v}
      remove from E' every edge incident on either u or v
return C
```

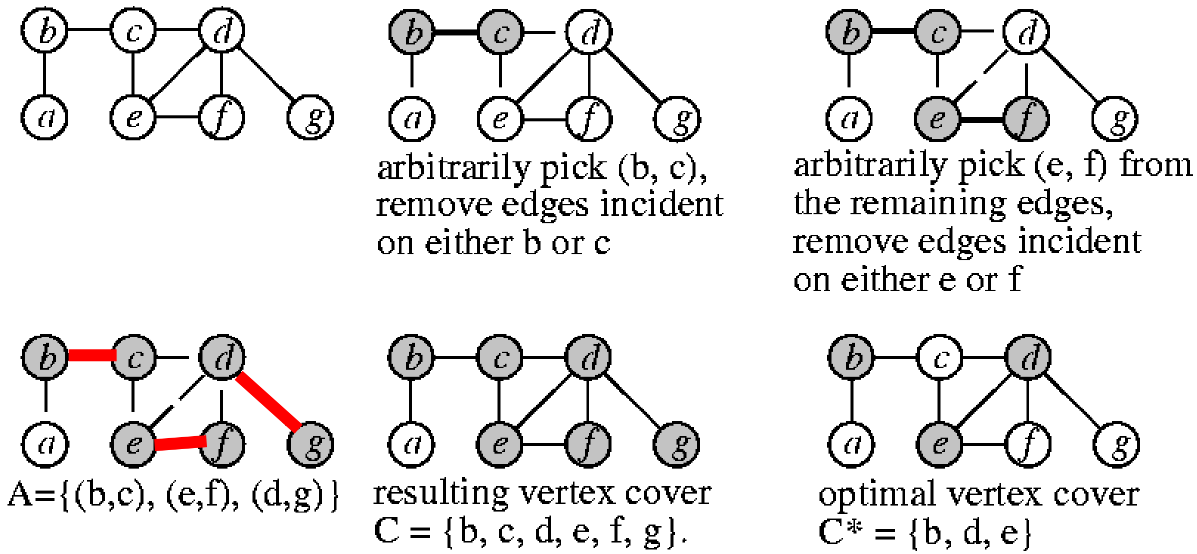
Time complexity: $O(E)$

|C|=6



|C*|=3

Approx-Vertex-Cover Has a Ratio Bound of 2

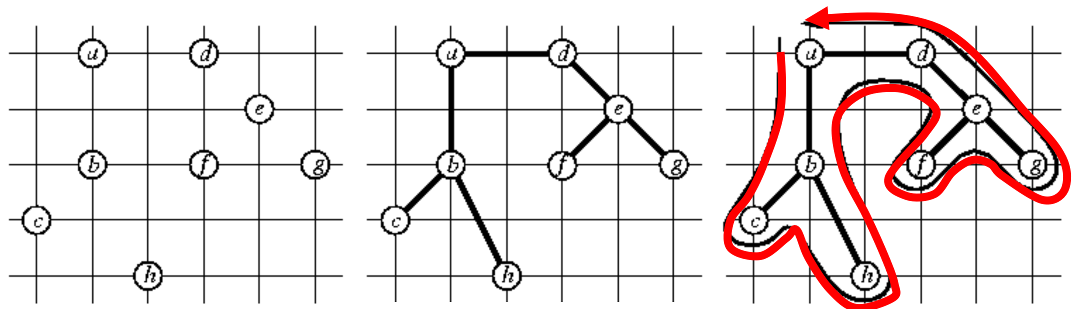


- Let A denote the set of edges picked in line 4 $\Rightarrow |C| = 2|A|$.
- Since no two edges in A share an endpoint, no two edges in A are covered by the same vertex from C^*.
- Every vertex should be covered by some vertex in $C^* \Rightarrow |A| \leq |C^*|$ and $|C| = 2|A| \leq 2|C^*|$.
- Recall: For a graph $G= (V, E)$, V' is a minimum vertex cover $\Leftrightarrow$ $V - V'$ is a maximum independent set.
  - Is there any polynomial-time approximation with a constant ratio bound for the maximum independent set problem?

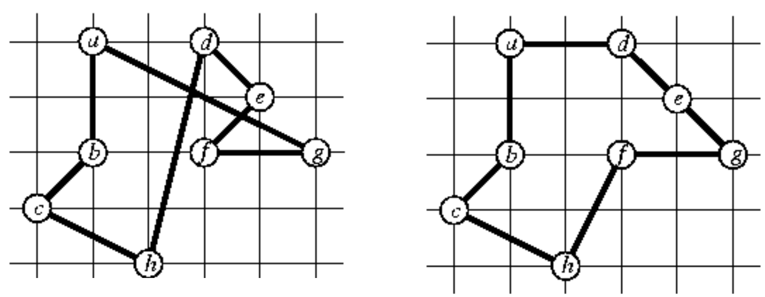
Approximation Algorithm for TSP

Approx-TSP-Tour(G)
1. select a vertex $r \in V[G]$ to be a "root" vertex;
2. grow a minimum spanning tree T for G from root r using MST-Prim(G, d, r)
3. let L be the list of vertices visited in a preorder tree walk of T
4. **return** the HC H that visits the vertices in the order L

- Time complexity: $O(V \lg V)$.



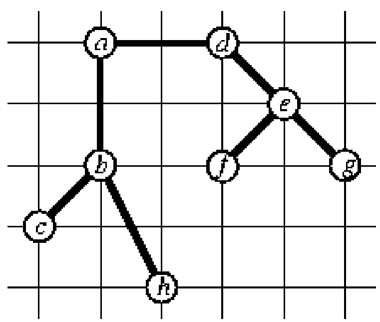
Run MST-Prim: *T* Preorder traversal of *T*



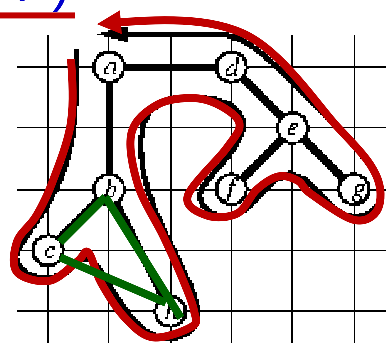
Resulting TSP tour *H* from the preorder traversal of *T* Optimal TSP tour *H**

Approx-TSP-Tour with Triangle Inequality

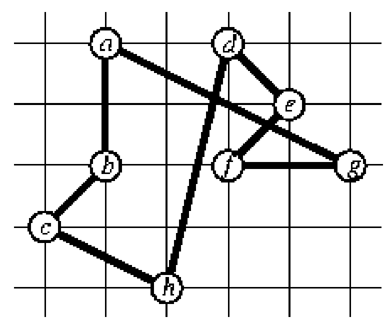
- Inter-city distances satisfy **triangle inequality** if for all vertices $u, v, w \in V$, $d(u, w) \leq d(u, v) + d(v, w)$.
- **Approx-TSP-Tour with triangle inequality has a ratio bound of 2.**
  - Let T = MST; Let H^*= optimal tour
  - H^* is formed by some tree plus an edge: $c(T) \leq c(T^*) \leq c(H^*)$
  - Let W = a full walk along T, e.g. $a,b,c,b,h,b,a,d,e,f,e,g,e,d,a$
  - Since W traverses every edge of T twice: $c(W) \leq 2 \times c(T)$
  - H = removed from W all but first visit to each vertex, e.g. a,b,c,h,d,e,f,g
  - Triangle inequality: removing vertex does not increase cost: $c(H) \leq c(W)$
  - so, $c(H) \leq 2 \times c(H^*)$



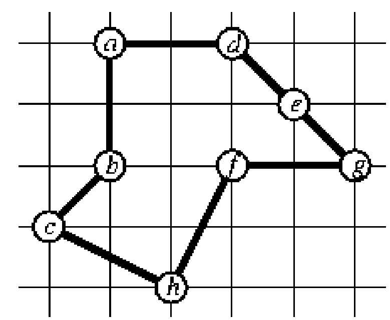
Run MST-Prim: T



Preorder traversal W of T



Resulting TSP tour H from the preorder traversal of T



Optimal TSP tour H^*

TSP without Triangle Inequality

- If P $\neq$ NP, there is no polynomial-time approximation algorithm with constant ratio bound ρ for the general TSP.
  - Suppose on the contrary that there is such an algorithm *A* with a constant ρ. We will use *A* to solve HC in polynomial time.
  - Algorithm for HC
    1. Convert $G = (V, E)$ into an instance G' of TSP with cities V (resulting in a complete graph $G' = (V, E')$):

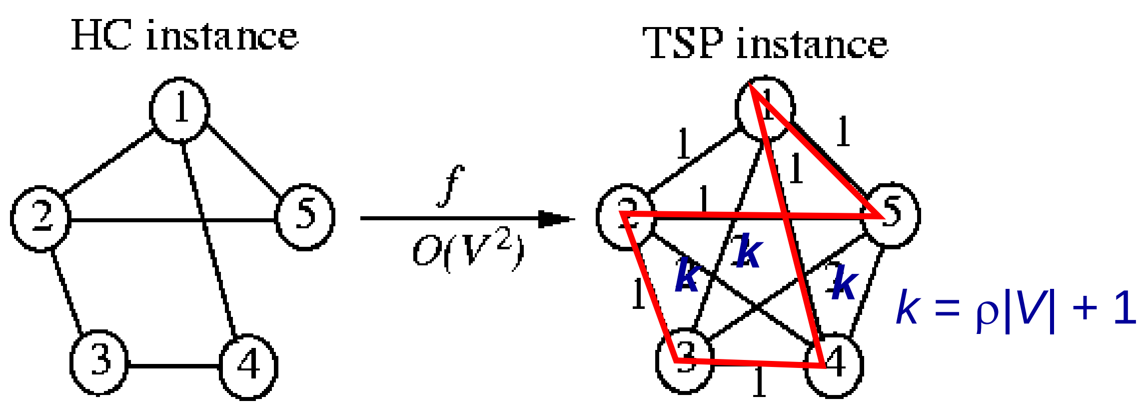
$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E , \\ \rho\,|V| + 1 & \text{otherwise .} \end{cases}$$

    2. Run *A* on G' with c
    3. If the reported cost $\leq \rho|V|$, then return "Yes" (i.e., *G* contains a tour that is an HC), else return "No."

Fact: HC $\leq_P$ TSP

Correctness

- If G has an HC: G' contains a tour of cost $|V|$ by picking edges in E, each with cost of 1.
- If G does not have an HC: any tour of G' must use some edge not in E, which has a total cost of $\geq$ $(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$.
- A guarantees to return a tour of cost $\leq \rho \times$ cost of an optimal tour if G contains an HC $\Rightarrow$ A returns a cost $\leq \rho|V|$ if G contains an HC; A returns a cost $> \rho|V|$, otherwise.
- A solve HC in polynomial time



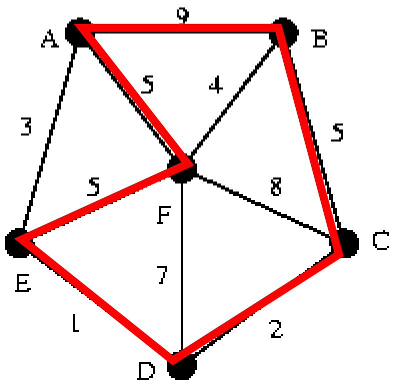
HC: <1, 5, 2, 3, 4, 1>

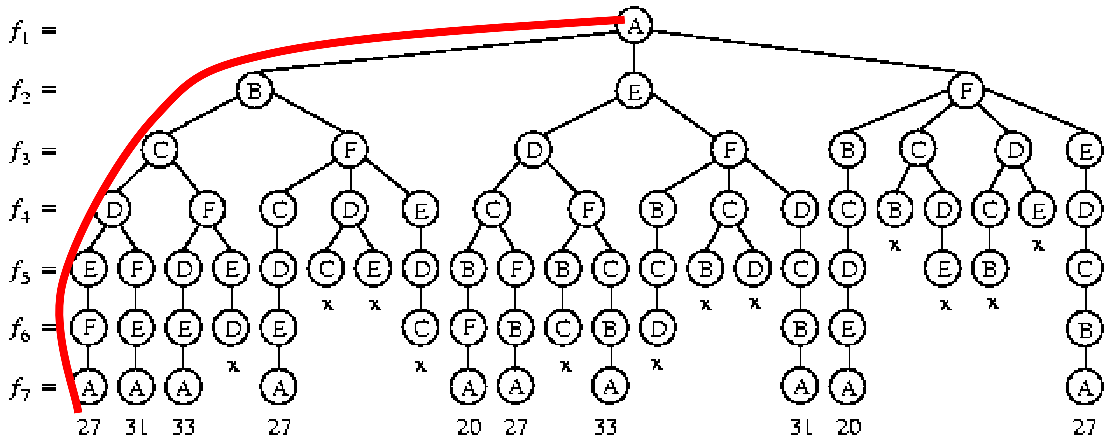
tour <1, 5, 2, 3, 4, 1> with distance bound B = $\rho|V|$

Skip

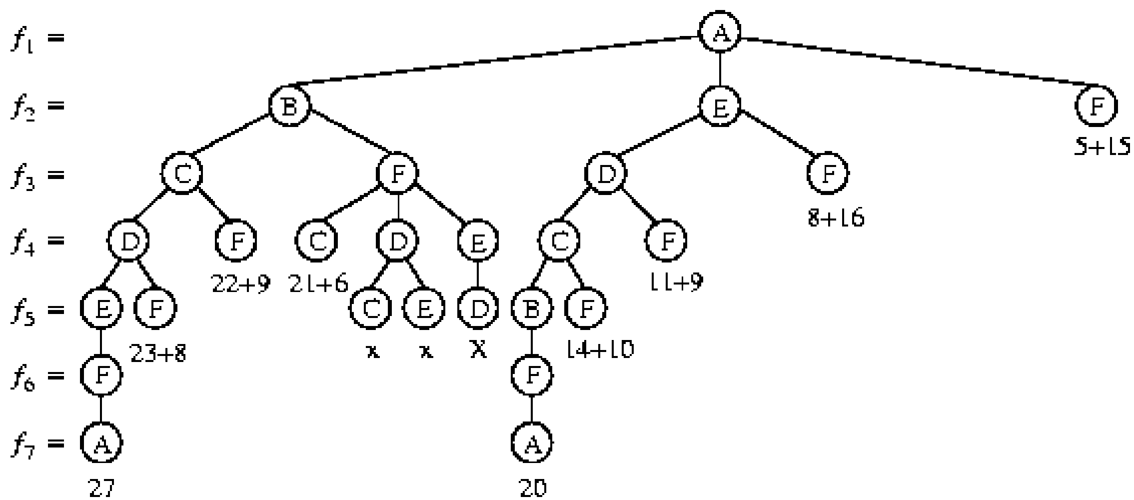
Exhaustive Search vs. Branch and Bound

- TSP example





Backtracking/exhaustive search

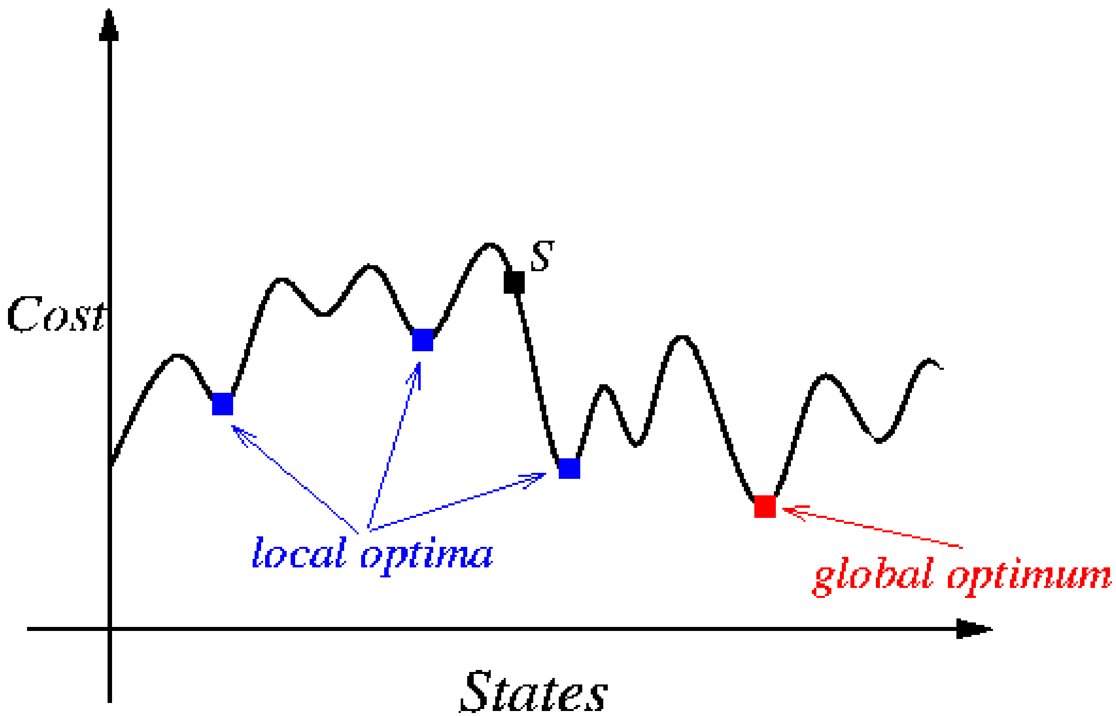


Branch and bound

Simulated Annealing

Skip

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- Chen and Chang, "Modern floorplanning based on fast simulated annealing," ISPD-05 (TCAD, April 2006).



Simulated Annealing Basics

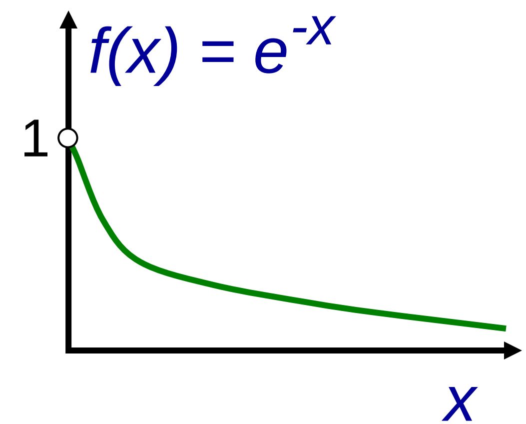
Skip

- Non-zero probability for "up-hill" moves.
- Probability depends on
  1. magnitude of the "up-hill" movement
  2. total search time

$p = \min\{1, e^{-\Delta C/T}\}$

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{ ``}down-hill\text{'' } moves */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{ ``}up-hill\text{'' } moves */ \end{cases}$$

- $\Delta C = cost(S') - Cost(S)$
- T: Control parameter (temperature)
- Annealing schedule: $T = T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$.
- Try a certain # of solutions for each temperature till "frozen."



Algorithmic Paradigms

Skip

Algorithmic Approaches

- Dynamic programming
- graph algorithm
- Greedy algorithm
- Branch & bound

Mathematical Programming

- Linear programming
- Integer linear prog.
- Quadratic prog.
- Nonlinear prog.

Nondeterministic Approaches

- Simulated annealing
- Machine learning
- Genetic algorithm
- Ant colony

Key Research Methodologies: CAR

Criticality

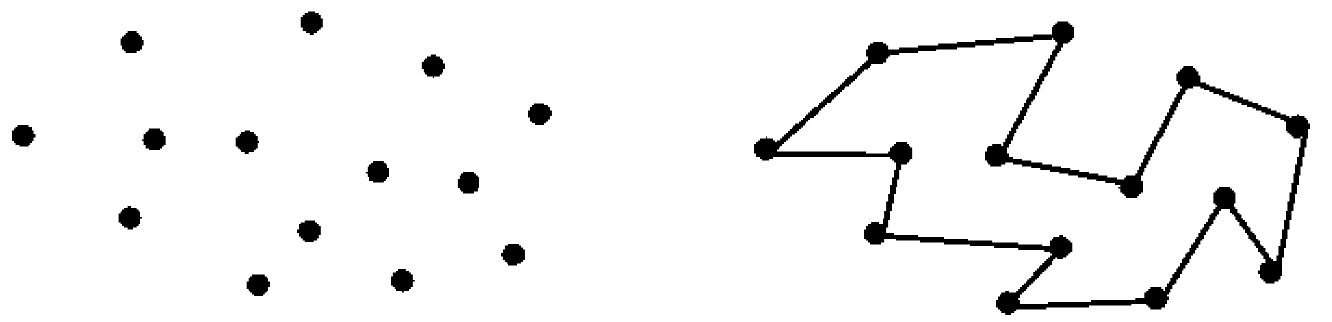
Abstraction

Restriction

Appendix

Verification Algorithm: TSP $\in$ NP

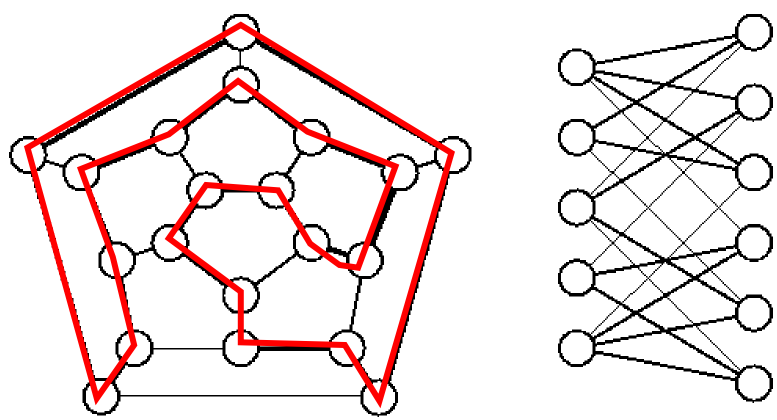
- **Verification algorithm:** a 2-argument algorithm *A*, where one argument is an input string *x* and the other is a binary string *y* (called a **certificate**). *A* verifies *x* if there exists *y* s.t. *A* answers "yes."
- Exp: Is TSP $\in$ NP?
- Need to **check** a TSP solution in polynomial time.
  - Guess a tour (certificate).
  - Check if the tour visits every city exactly once.
  - Check if the tour returns to the start.
  - Check if total distance $\leq B$.
- All can be done in $O(n)$ time, so TSP $\in$ NP.



Polynomial Reduction: HC $\leq_P$ TSP

- **The Hamiltonian Circuit Problem (HC)**
  - **Instance:** an undirected graph $G = (V, E)$.
  - **Question:** is there a cycle in G that includes every vertex exactly once?
- **TSP: The Traveling Salesman Problem**
- **Claim:** HC $\leq_P$ TSP.
  1. Define a function f mapping **any** HC instance into a TSP instance, and show that ***f* can be computed in polynomial time**.
  2. Prove that G has an HC iff the reduced instance has a TSP tour **with distance $\leq$ *B*** ($x \in$ HC $\Leftrightarrow f(x) \in$ TSP).

Hamiltonian



Hamiltonian   nonhamiltonian

nonhamiltonian

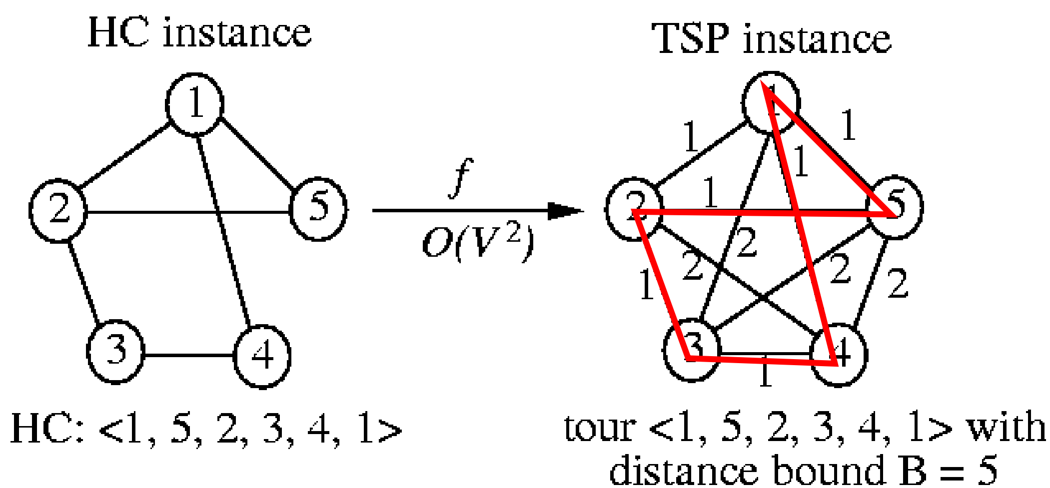
HC $\leq_P$ TSP: Step 1

1. Define a reduction function *f* for HC $\leq_P$ TSP.
   - Given an HC instance $G = (V, E)$ with n vertices
     - Create a set of n cities labeled with names in V.
     - Assign distance between u and v

$$d(u, v) = \begin{cases} 1, & \text{if } (u, v) \in E, \\ 2, & \text{if } (u, v) \notin E. \end{cases}$$

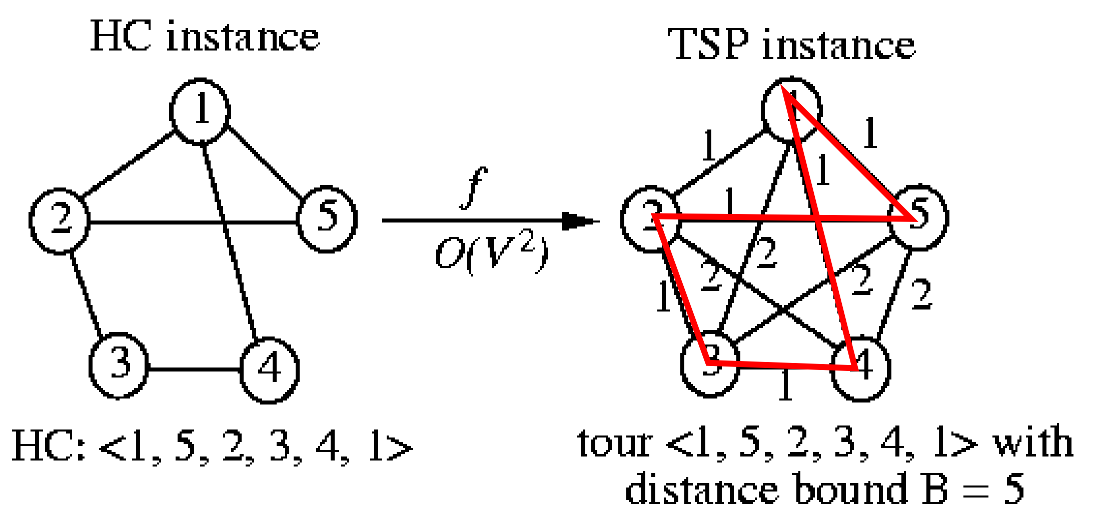
     - Set bound $B = n$.
   - f can be computed in $O(V^2)$ time.

look for the difference between the two problems to make the reduction!!



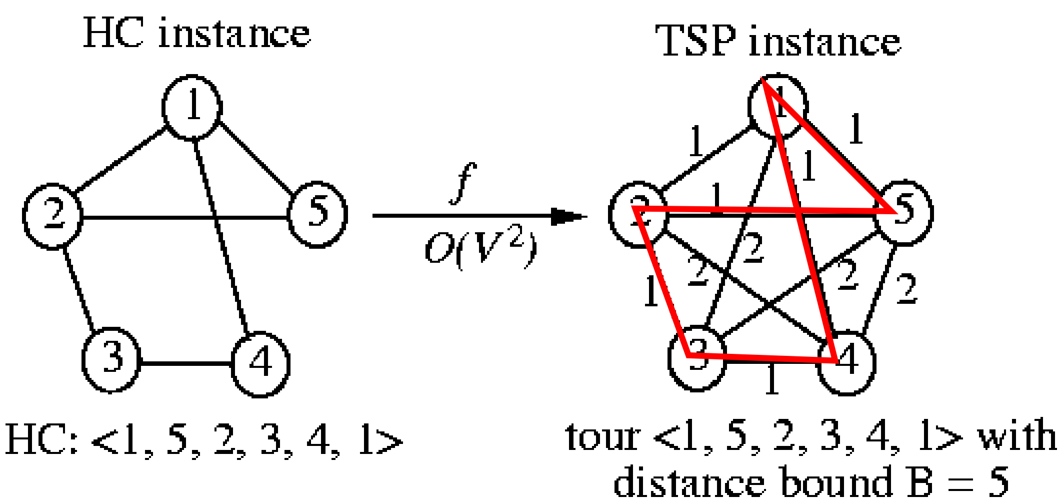
HC $\leq_P$ TSP: Step 2

2. *G* has an HC iff the reduced instance has a TSP **with distance ≤ *B***.
   - $x \in$ HC $\Rightarrow f(x) \in$ TSP.
     - Suppose the HC is $h = \langle v_1, v_2, \ldots, v_n, v_1 \rangle$. Then, *h* is also a tour in the transformed TSP instance.
     - The distance of the tour *h* is $n = B$ since there are *n* consecutive edges in *E*, and so has distance 1 in $f(x)$.
     - Thus, $f(x) \in$ TSP ($f(x)$ has a TSP tour with distance $\leq B$).



HC $\leq_P$ TSP: Step 2 (cont'd)

2. *G* has an HC iff the reduced instance has a TSP **with distance ≤ *B***.
   - $f(x) \in$ TSP $\Rightarrow x \in$ HC.
     - Suppose there is a TSP tour **with distance ≤ *n* = *B***. Let it be $<v_1, v_2, \ldots, v_n, v_1>$..
     - Since **distance of the tour ≤ *n*** and **there are *n* edges** in the TSP tour, the tour contains only edges in *E* since all edge weights are equal to 1.
     - Thus, $<v_1, v_2, \ldots, v_n, v_1>$ is a Hamiltonian cycle ($x \in$ HC).

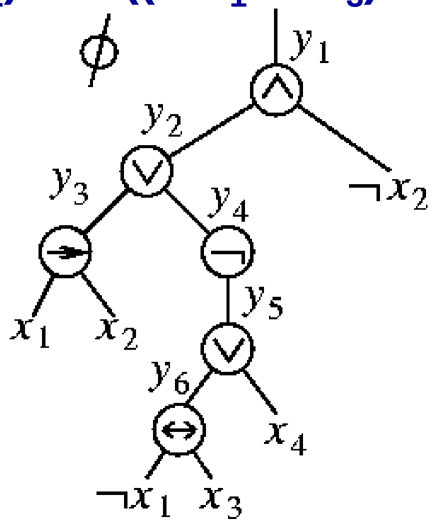


3SAT is NP-Complete

- **3SAT:** Satisfiability of boolean formulas in **3-conjunctive normal form (3-CNF)**.
  - Each clause has exactly 3 distinct literals, e.g.,

    $\phi = (x_1 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$
- **3SAT $\in$ NP** (will omit this part for other proofs).
- **3SAT is NP-hard:** SAT $\leq_P$ 3SAT.
  1. Construct a binary "parse" tree for input formula ϕ and introduce a variable y_i for the output of each internal node.

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

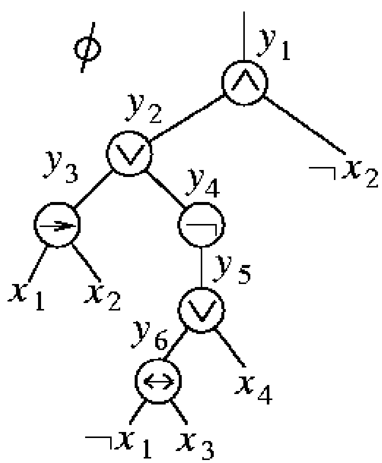


3SAT is NP-Complete (cont'd)

2. Rewrite ϕ as the AND of the root variable and a conjunction of clauses describing the operation of each node.

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

3. Convert each clause ϕ'_i into CNF.
   - Construct the **disjunctive normal form** for $\neg \phi'_i$ and then apply DeMorgan's law to get the CNF formula ϕ''_i.
   - E.g., $\neg \phi'_1 = \neg (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) =$ $(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$
   - $\phi''_1 = \neg(\neg \phi'_1) = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$.



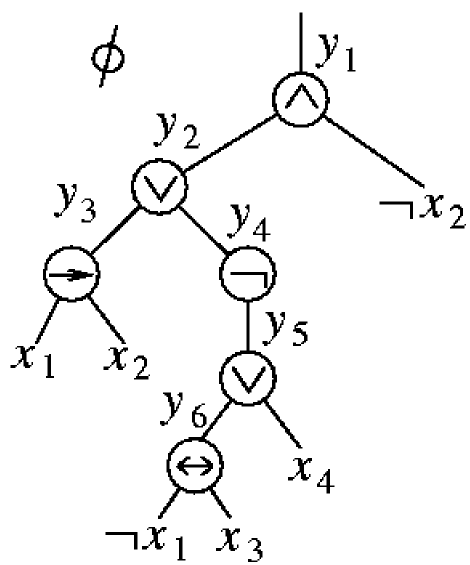
y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	**1**	**1**	**0**
1	1	0	1
1	**0**	**1**	**0**
1	**0**	**0**	**0**
0	1	1	1
0	**1**	**0**	**0**
0	0	1	1
0	0	0	1

truth table for $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$

3SAT is NP-Complete (cont'd)

4. Make each clause C_i have **exactly** 3 distinct literals to get ϕ'''.
   - C_i has 3 distinct literals: do nothing.
   - C_i has 2 distinct literals:
     $\boldsymbol{C_i = (l_1 \vee l_2) = (l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)}$.
   - C_i has only 1 literal: $\boldsymbol{C_i = l =}$
     $\boldsymbol{(l \vee p \vee q) \wedge (l \vee \neg p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee \neg q)}$.

- **Claim:** The 3-CNF formula ϕ''' is satisfiable $\Leftrightarrow$ ϕ is satisfiable.
- All transformations can be done in polynomial time.



y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	**1**	**1**	**0**
1	1	0	1
1	**0**	**1**	**0**
1	**0**	**0**	**0**
0	1	1	1
0	**1**	**0**	**0**
0	0	1	1
0	0	0	1

truth table for $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$