南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

(slides edited from Prof. Shiqi Yu)

NP and Computational Intractability

Algorithm Design Patterns and Antipatterns

- **Algorithm design patterns.**
  - Greedy
  - Divide and conquer
  - Dynamic programming
  - Duality (e.g., network flow)
  - Randomization
  - Reductions
- **Algorithm design antipatterns.**
  - **NP**-completeness. Polynomial-time algorithm unlikely.
  - **PSPACE**-completeness. Polynomial-time certification algorithm unlikely.
  - Undecidability. No algorithm possible.

1. Poly-Time Reductions

Classify Problems by Computational Requirements

- **Q.** Which problems will we be able to solve in practice?
- **A working definition.** Those with polynomial-time algorithms.

von Neumann (1953) | Nash (1955) | Gödel (1956) | Cobham (1964) | Edmonds (1965) | Rabin (1966)

- **Theory.** Definition is broad and robust.

Turing machine, word RAM, uniform circuits, …

- **Practice.** Poly-time algorithms scale to huge problems.

constants tend to be small, e.g., $3n^2$

Classify Problems by Computational Requirements

- **Q.** Which problems will we be able to solve in practice?
- **A working definition.** Those with polynomial-time algorithms.

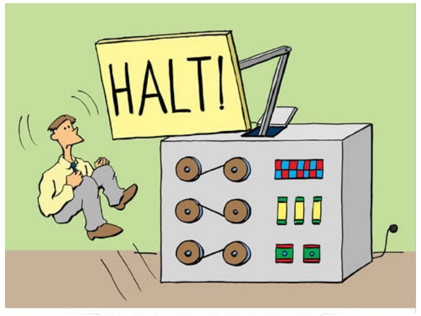
Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring
Linear programming	Integer linear programming

Classify Problems

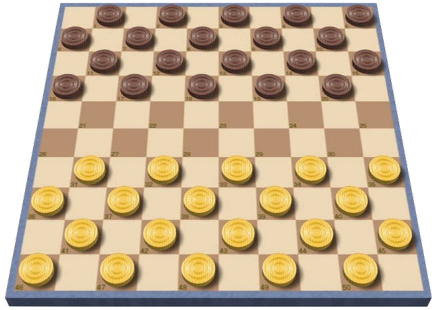
- **Goal.** Classify problems according to those that can be solved in polynomial-time and those that cannot.
- **Provably requires exponential-time:**

input size: c + log k

  - Given a Turing machine, does it halt in at most *k* steps?
  - Given a board position in an *n*-by-*n* generalization of checkers, can black guarantee a win?



Alan designed the perfect computer



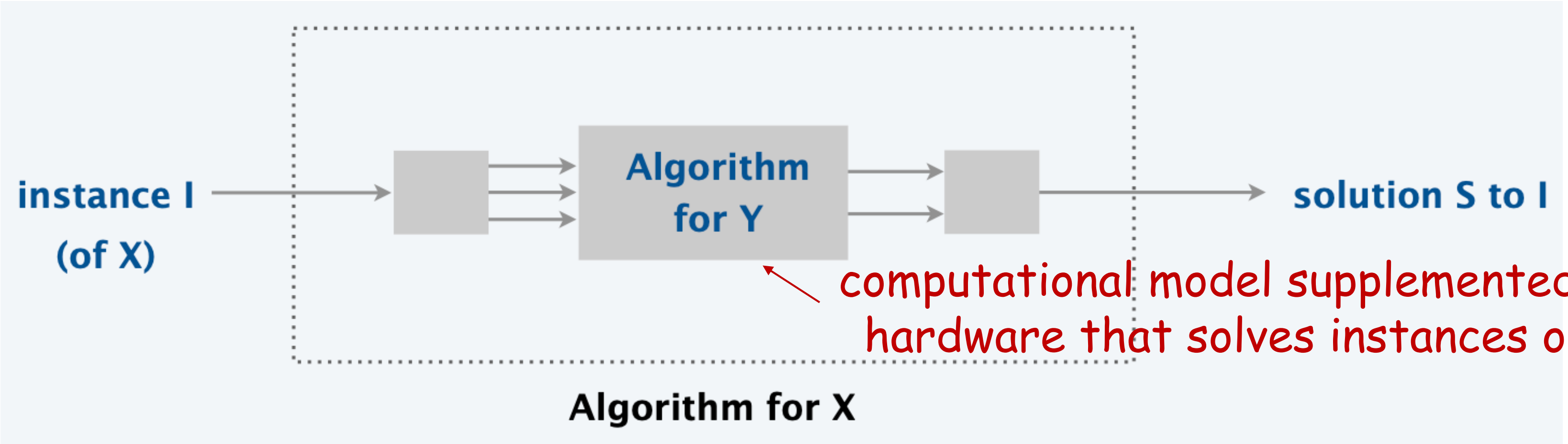
- **Frustrating news.** Huge number of fundamental problems have defied classification for decades.

Polynomial-Time Reductions

- **Q.** Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?
- **Reduction.** Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:
  - Polynomial number of standard computational steps, plus
  - Polynomial number of calls to oracle that solves problem Y.

don't confuse with "reduce from"

can be viewed as a magic black box



computational model supplemented by special piece of hardware that solves instances of Y in a single step

Polynomial-Time Reductions

- **Q.** Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?
- **Reduction.** Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:
  - Polynomial number of standard computational steps, plus
  - Polynomial number of calls to oracle that solves problem Y.

don't confuse with "reduce from"

can be viewed as a magic black box

Karp reduction allows only one oracle call

- **Notation.** $X \leq_P Y$.
- **Note.** We pay for time to write down instances of Y sent to black box $\Rightarrow$ instances of Y must be of polynomial size.

Polynomial-Time Reductions

- **Design algorithms.** If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

- **Establish intractability.** If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

up to cost of reduction

- **Establish equivalence.** If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time iff Y can be.

- **Bottom line.** Reductions classify problems according to relative difficulty.

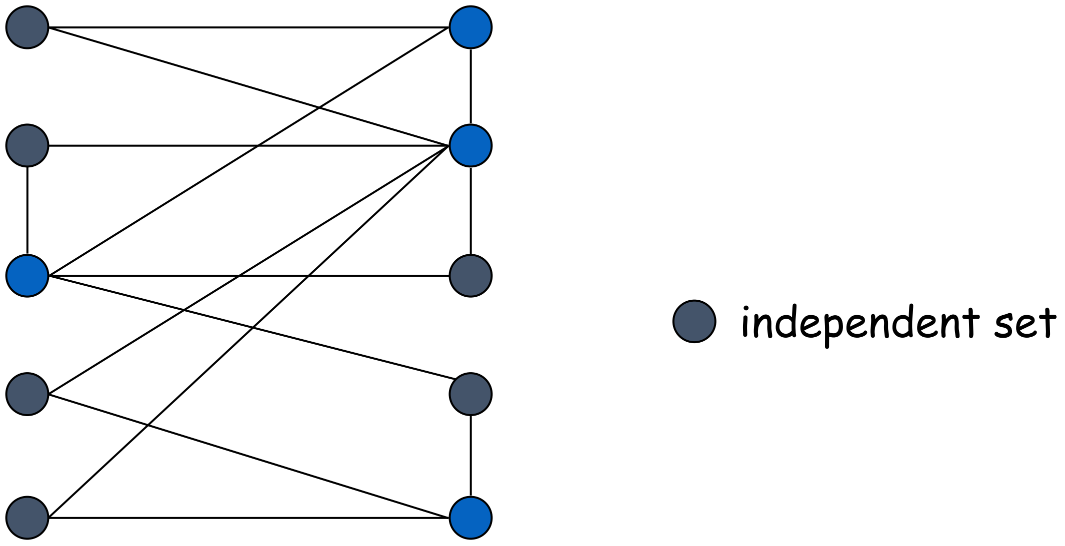
2. Reduction By Simple Equivalence

Basic reduction strategies:

- Reduction by simple equivalence
- Reduction from special case to general case
- Reduction via "gadgets"

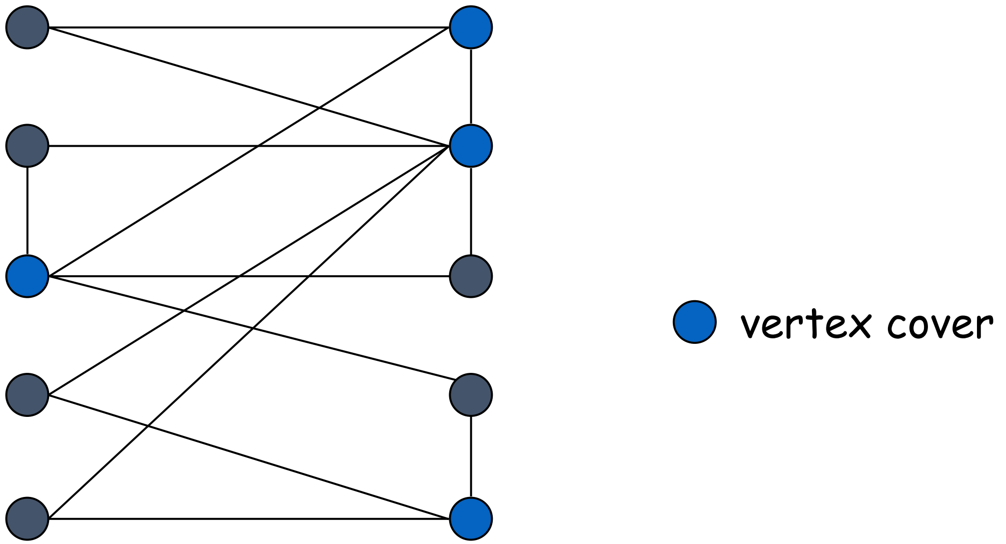
Independent Set

- **INDEPENDENT-SET.** Given a graph G = (V, E) and an integer k, is there a subset S of k (or more) vertices such that no two in S are adjacent?
- **Ex.** Is there an independent set of size ≥ 6? Yes.
- **Ex.** Is there an independent set of size ≥ 7? No.



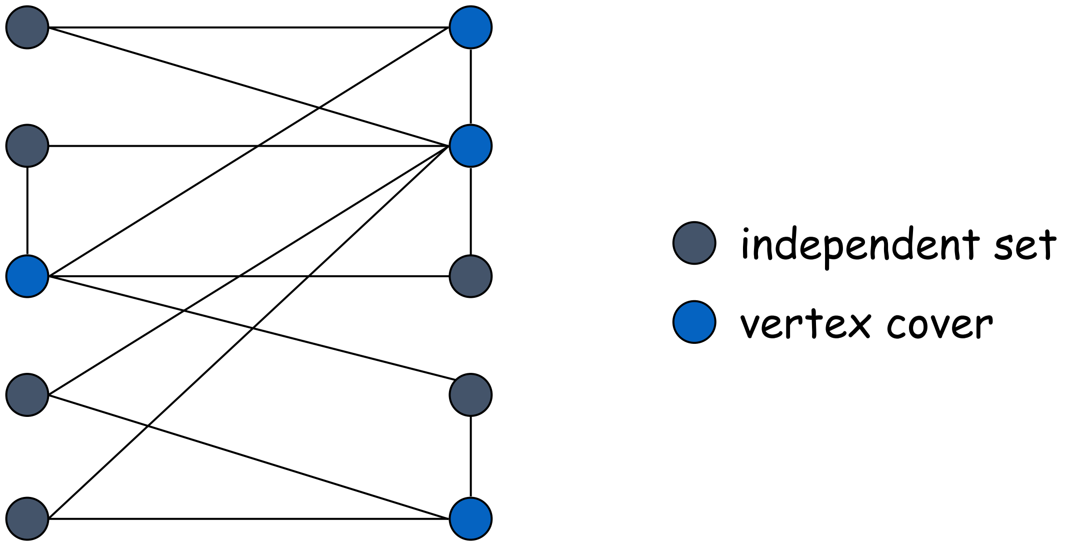
Vertex Cover

- **VERTEX-COVER.** Given a graph G = (V, E) and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?
- **Ex.** Is there a vertex cover of size ≤ 4? Yes.
- **Ex.** Is there a vertex cover of size ≤ 3? No.



Vertex Cover and Independent Set

- **Claim.** INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- **Pf.** We show S is an independent set iff V – S is a vertex cover.



Vertex Cover and Independent Set

- **Claim.** INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- **Pf.** We show S is an independent set iff V – S is a vertex cover.

$\Rightarrow$: Let S be any independent set.

- Consider an arbitrary edge $(u, v) \in E$.
- S independent $\Rightarrow u \notin S$ or $v \notin S \Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, V – S covers (u, v).

$\Leftarrow$: Let V – S be any vertex cover.

- Consider an arbitrary edge $(u, v) \in E$.
- V – S vertex cover $\Rightarrow u \in V - S$ or $v \in V - S \Rightarrow u \notin S$ or $v \notin S$.
- Thus, S is an independent set. ▪

3. Reduction from Special Case to General Case

Basic reduction strategies:

- Reduction by simple equivalence
- Reduction from special case to general case
- Reduction via "gadgets"

Set Cover

- **SET-COVER.** Given a set U of elements, a collection of subsets of U, and an integer k, are there ≤ k of these subsets whose union is equal to U?
- **Sample application.**
  - m available pieces of software.
  - Set U consists of n capabilities that we would like our system to have.
  - The i-th piece of software provides the subset $S_i \subseteq U$ of capabilities.
  - **Goal:** achieve all n capabilities using fewest pieces of software.
- **Ex.**

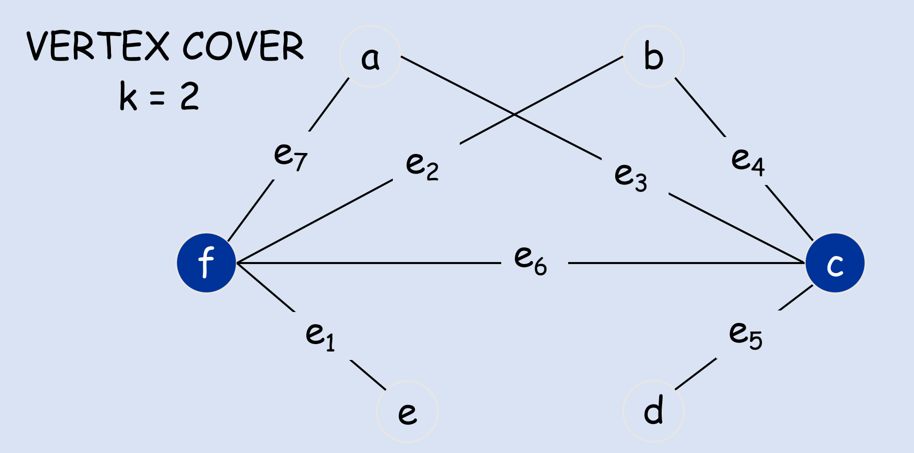
$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$k = 2$

$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

Vertex Cover Reduces to Set Cover

- **Claim.** VERTEX-COVER $\leq_P$ SET-COVER.
- **Pf.** Given a VERTEX-COVER instance G = (*V*, *E*) and *k*, we construct a SET-COVER instance whose size equals the size of the vertex cover instance.
- **Construction.** Create SET-COVER instance (*U*, *S*, *k*):
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$, $S = \{S_v : v \in V\}$



SET COVER
k = 2

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
$S_a = \{3, 7\}$ $S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$ $S_d = \{5\}$
$S_e = \{1\}$ $S_f = \{1, 2, 6, 7\}$

Vertex Cover Reduces to Set Cover

- **Claim.** VERTEX-COVER $\leq_P$ SET-COVER.
- **Pf.** Given a VERTEX-COVER instance G = (*V*, *E*) and *k*, we construct a SET-COVER instance whose size equals the size of the vertex cover instance.
- **Construction.** Create SET-COVER instance (*U*, *S*, *k*):
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$, $S = \{S_v : v \in V\}$
- **Lemma.** G = (V, E) contains a vertex cover of size *k* iff (*U, S, k*) contains a set cover of size *k*.
- **Pf.** Let $X \subseteq V$ be a vertex cover of size *k*; let $Y \subseteq S$ be a set cover of size *k*.
  - $\Rightarrow$: $Y = \{ S_v : v \in X \}$ is a set cover of size *k*.
  - $\Leftarrow$: $X = \{ v : S_v \in Y \}$ is a vertex cover of size *k*. ▪

4. Reductions via "Gadgets"

Basic reduction strategies:

- Reduction by simple equivalence
- Reduction from special case to general case
- Reduction via “gadgets”

Satisfiability

- **Q.** Given a propositional formula Φ, is there a truth assignment to its variables such that Φ = 1, i.e., is there a satisfying truth assignment?
- **Ex.**

a	b	c	$(a \wedge b) \vee c$	$(a \wedge \neg a) \vee (c \wedge \neg c)$
			yes	no
1	1	1	1	0
0	1	1	1	0
0	0	1	1	0
0	1	0	0	0
1	0	1	1	0
1	0	0	0	0
1	0	1	1	0
0	0	0	0	0

Satisfiability

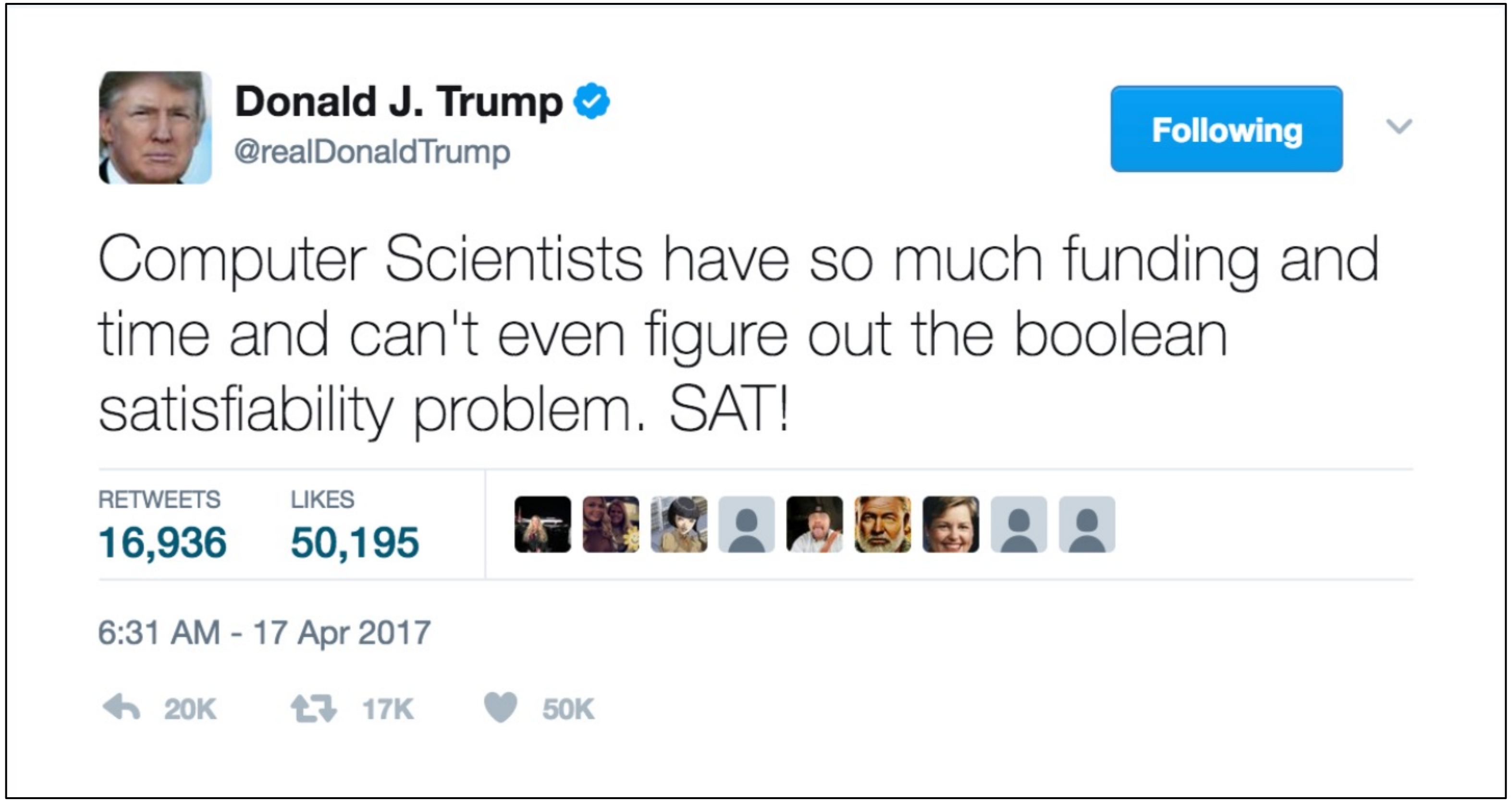
- **Literal.** A Boolean variable or its negation. x_i or $\overline{x_i}$
- **Clause.** A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$
- **Conjunctive normal form (CNF).** A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$
- **SAT.** Given a CNF formula Φ, does it have a satisfying truth assignment?
- **3-SAT.** SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Ex: $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false.

Satisfiability is Hard

- **Hypothesis.** There does not exist a poly-time algorithm for 3-SAT.
- **P vs. NP.** This hypothesis is equivalent to **P ≠ NP** conjecture.

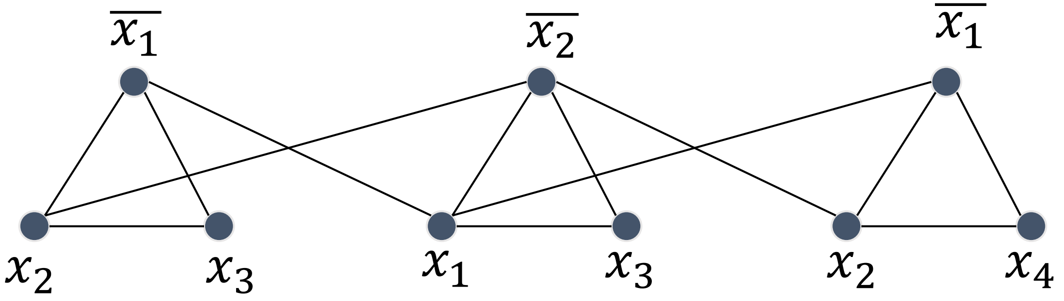


3-Satisfiability Reduces to Independent Set

- **Claim.** 3-SAT $\leq_P$ INDEPENDENT-SET.
- **Pf.** Given a 3-SAT instance Φ, construct an INDEPENDENT-SET instance (*G*, *k*) that has an independent set of size $k = |\Phi|$ iff Φ is satisfiable.

  number of clauses
- **Construction.**
  - *G* contains 3 vertices for each clause, one for each literal.
  - Connect 3 literals in a clause in a triangle.
  - Connect literal to each of its negations.

G



k = 3 $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

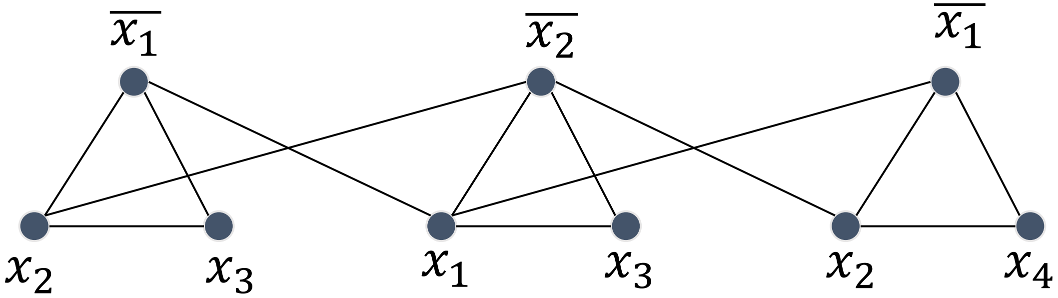
3-Satisfiability Reduces to Independent Set

- **Lemma.** Φ is satisfiable iff G contains independent set of size $k = |\Phi|$.
- **Pf.**

$\Rightarrow$: Consider any satisfying assignment for Φ.

- Choose one true literal from each clause/triangle.
- No two literals chosen in one triangle; complementary literals not both chosen.
- This is an independent set of size $k = |\Phi|$.

G



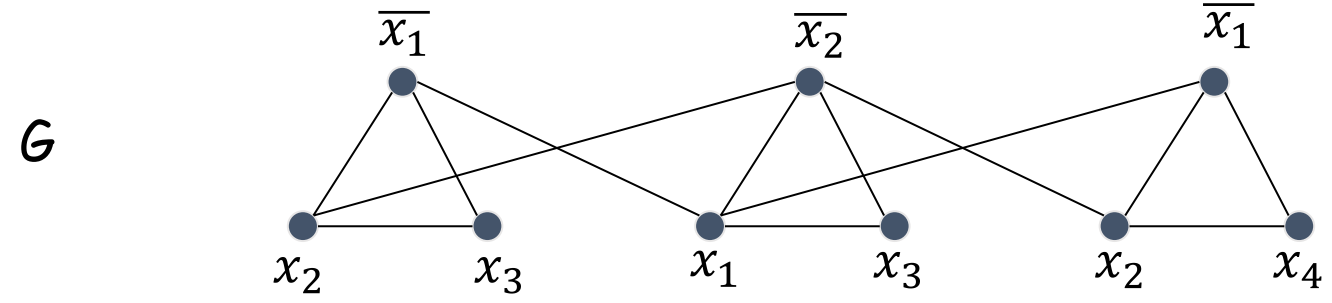
k = 3 $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

3-Satisfiability Reduces to Independent Set

- **Lemma.** Φ is satisfiable iff G contains independent set of size $k = |\Phi|$.
- **Pf.**

$\Leftarrow$: Let S be an independent set of size $k = |\Phi|$.

- S must contain exactly one vertex in each triangle.
- Set these literals to true (and remaining literals consistently).
- All clauses in Φ are satisfied. ▪



k = 3 $\quad \Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

Summary

- **Basic reduction strategies.**
  - Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
  - Special case to general case: VERTEX-COVER $\le_P$ SET-COVER.
  - Encoding with gadgets: 3-SAT $\le_P$ INDEPENDENT-SET.

- **Transitivity.** If X $\le_P$ Y and Y $\le_P$ Z, then X $\le_P$ Z.
- **Pf idea.** Compose the two algorithms.

- **Ex.** 3-SAT $\le_P$ INDEPENDENT-SET $\le_P$ VERTEX-COVER $\le_P$ SET-COVER.

Exercise: Three Types of Problems

- **Decision problem.** Does there exist a vertex cover of size ≤ k?
- **Search problem.** Find a vertex cover of size ≤ k.
- **Optimization problem.** Find a vertex cover of minimum size.

- **Goal.** Show that all three problems poly-time reduce to one another.