Slido: #ADA2022

CSIE 2136 Algorithm Design and Analysis, Fall 2022

NP Completeness - II

Hsu-Chun Hsiao

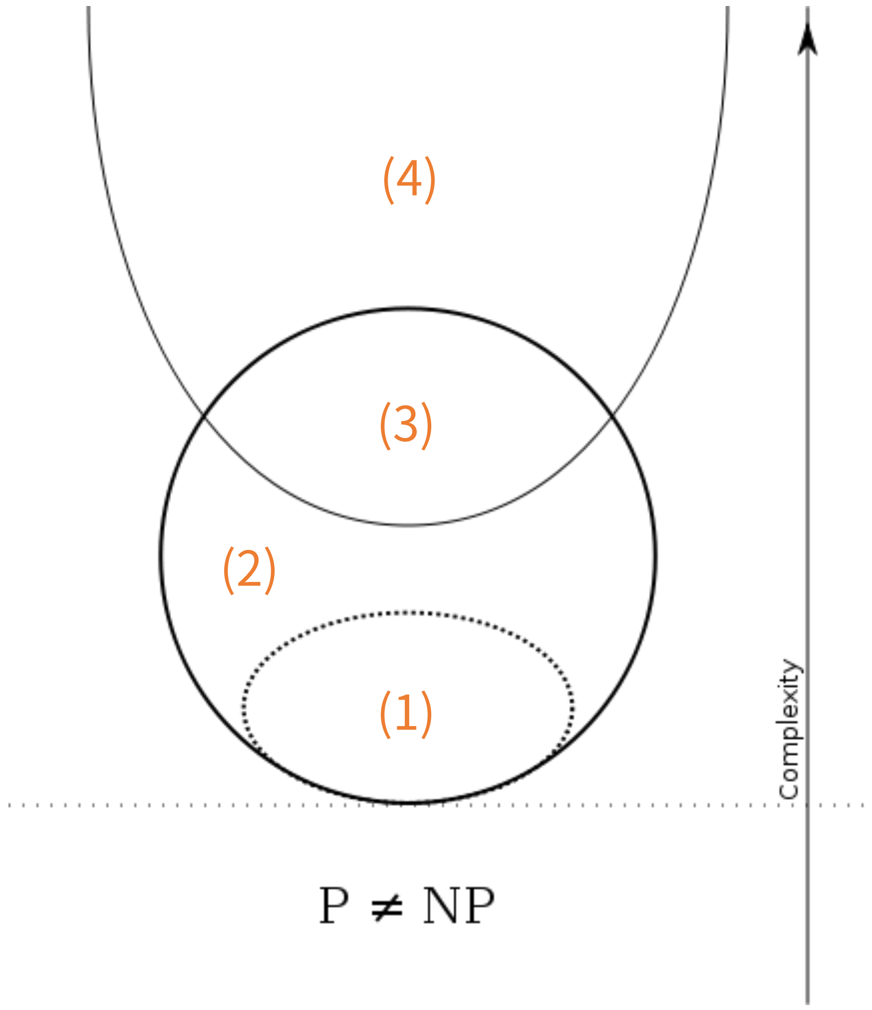
Agenda

- Review
- Brief explanation of polynomial-time solving vs. verification
- Proving NP-Completeness
- Handling NP-completeness by solvers

- Next week: approximation algorithms, more examples, & final review

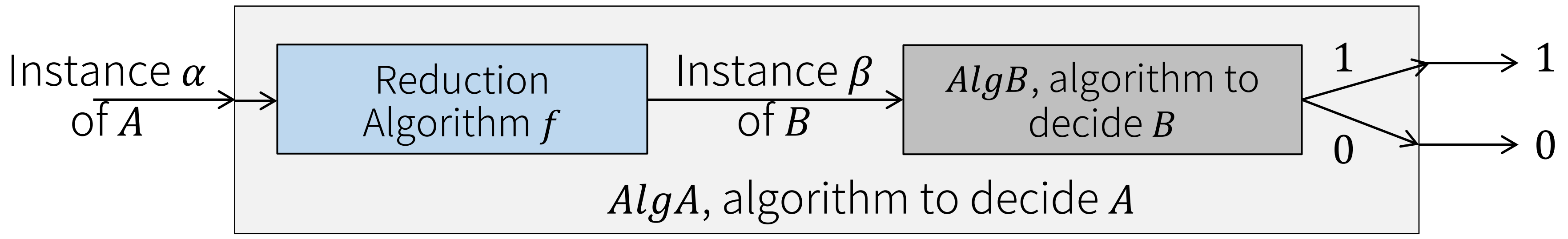
Review

P, NP, NP-Hard, NP-Complete



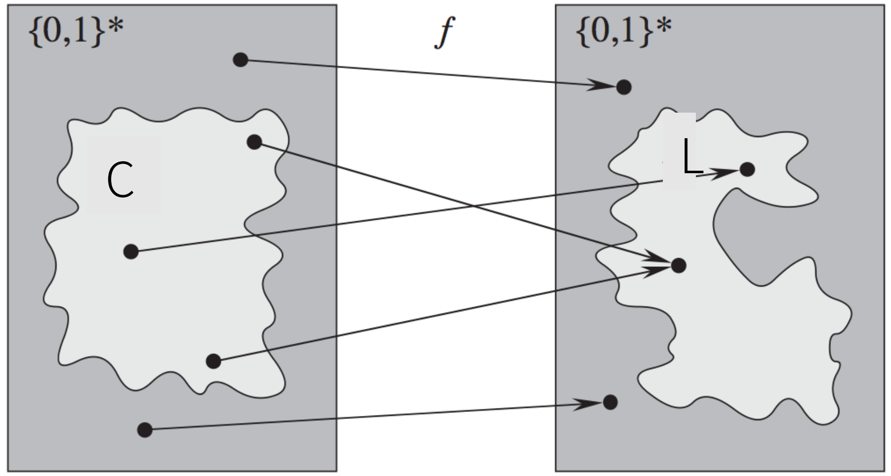
Polynomial-time reduction

- A polynomial-time reduction ($A \leq_p B$) is a polynomial-time algorithm for transforming every instance of a problem A into an instance of another problem B
  - Can help determine the hardness relationship between problems (within a polynomial-time factor)
  - $A \leq_p B$ implies A is no harder than B; equivalently, B is at least at hard as A



Proving NP-Completeness

$L \in$ NP-Complete $\Leftrightarrow L \in$ NP and $L \in$ NP-hard



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

1. Prove $L \in$ NP
2. Prove $L \in$ NP-hard ($C \le_p L$)
   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

Common misconceptions

Wrong	Correct
❌ f does not transform every problem A's instance.	✅ f must transform every problem A's instance.
❌ $f(\alpha)$ must cover every problem B's instance.	✅ $f(\alpha)$ may be a subset of problem B's instances.
❌ $A \leq_p B$ implies $B \leq_p A$.	✅ Reduction is directional; $A \leq_p B$ does not imply $B \leq_p A$.
❌ To prove $A \leq_p B$, we only need to show $AlgA(\alpha) = 1$ implies $AlgB(f(\alpha)) = 1$.	✅ While reduction is directional, the proof of correctness must be done both directions. That is, $AlgA(\alpha) = 1$ if and only if $AlgB(f(\alpha)) = 1$.
❌ If A can be reduced to B in $O(n)$ and there is an algorithm $AlgB$ for B runs in $O(n^3)$, then we can construct an algorithm for A runs in $O(n^3)$.	✅ We need to take into account possible size increase after f. If A can be reduced to B in $O(n)$ and the size increases to $O(n^2)$, and there is an algorithm $AlgB$ for B runs in $O(n^3)$, then we can construct an algorithm for A runs in $O(n^6)$.

Polynomial-time Solving & Polynomial-time Verification

Chap. 34.1, 34.2

Abstract problems

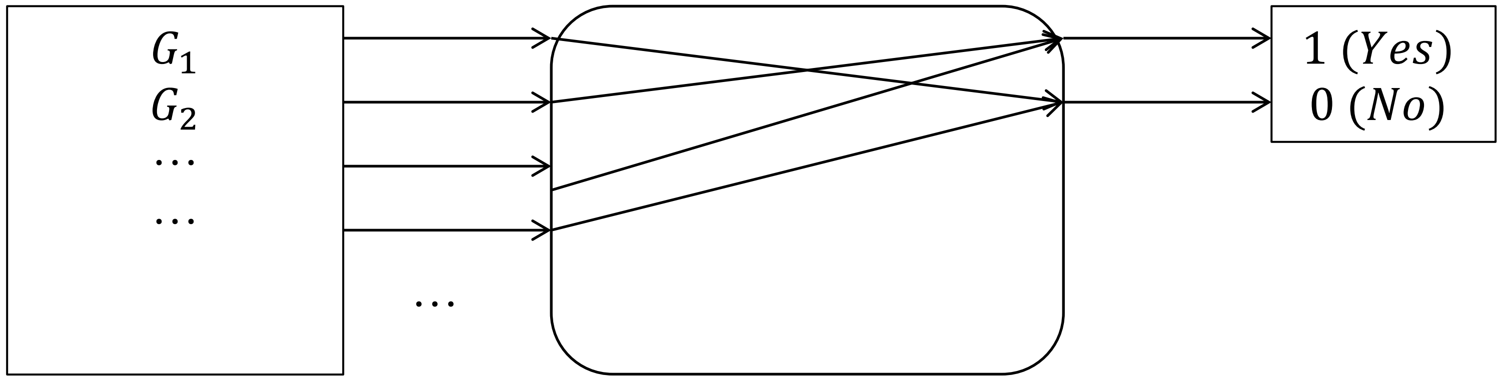
- I: the set of problem instances
- S: the set of problem solutions
- Q: an abstract problem, defined as a binary relation on I and S

Example of a decision problem, HAM-CYCLE:

I: $< G = (V, E) >$
所有可能的圖

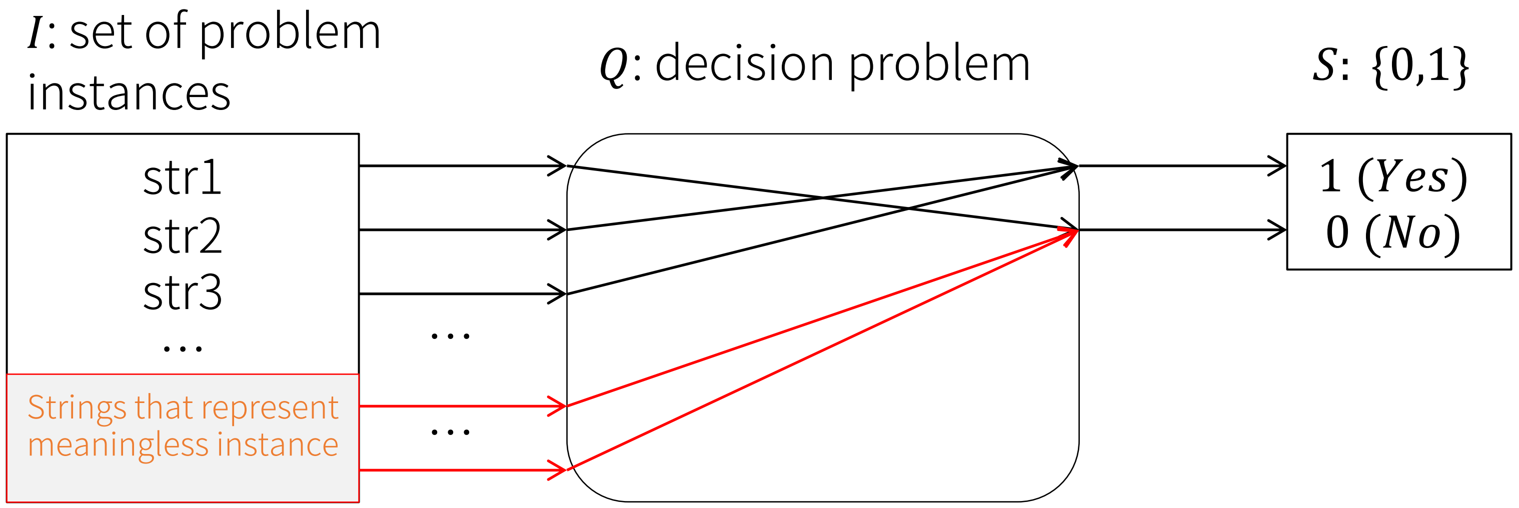
Q: HAM-CYCLE
是否存在 Hamiltonian Cycle?

S: $\{0,1\}$

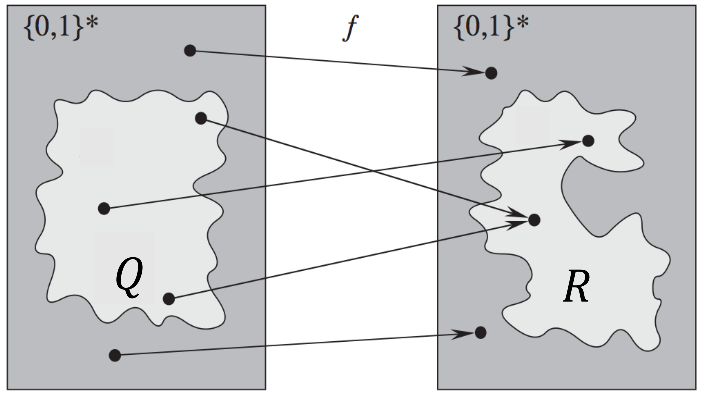


Representation of a decision problem

- I = the set of problem instances = Σ^*, where $\Sigma = \{0, 1\}$
- Q = a decision problem = $\{x \in \Sigma^*\colon\ Q(x) = 1\}$
  - 以答案為 1 的 instances 定義 decision problem Q
  - $Q = \{str2, str3\}$ in the example below
- Q can be reduced to R means $x \in Q \Leftrightarrow f(x) \in R$

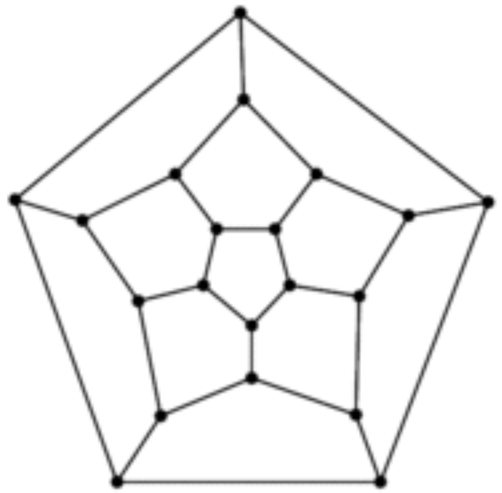


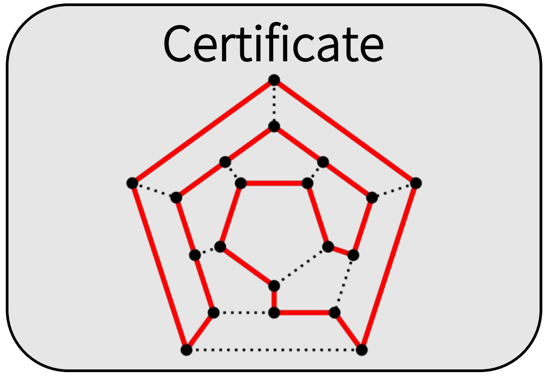
I: set of problem instances
Q: decision problem
S: $\{0,1\}$
str1
str2
str3
…
Strings that represent meaningless instance
…
…
1 (Yes)
0 (No)



Polynomial-time verification

- 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?

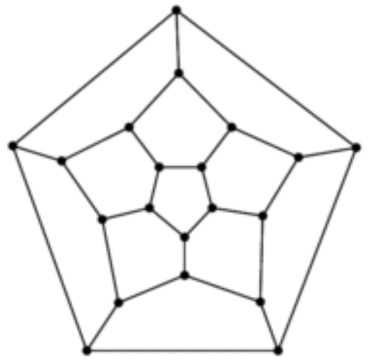




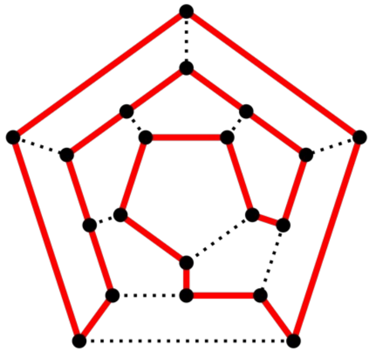
Verification algorithms

- Q = a decision problem = $\{x \in \Sigma^* \colon Q(x) = 1\}$
  - HAM-CYCLE = $\{\langle G \rangle \mid G$ has a Hamiltonian cycle$\}$
- Verification algorithm verifies x is in Q with the help of a certificate y

Input x



Certificate y



Verification algorithm: Is y a Hamiltonian cycle in the graph (encoded in x)?

YES

(x is in HAM-CYCLE)

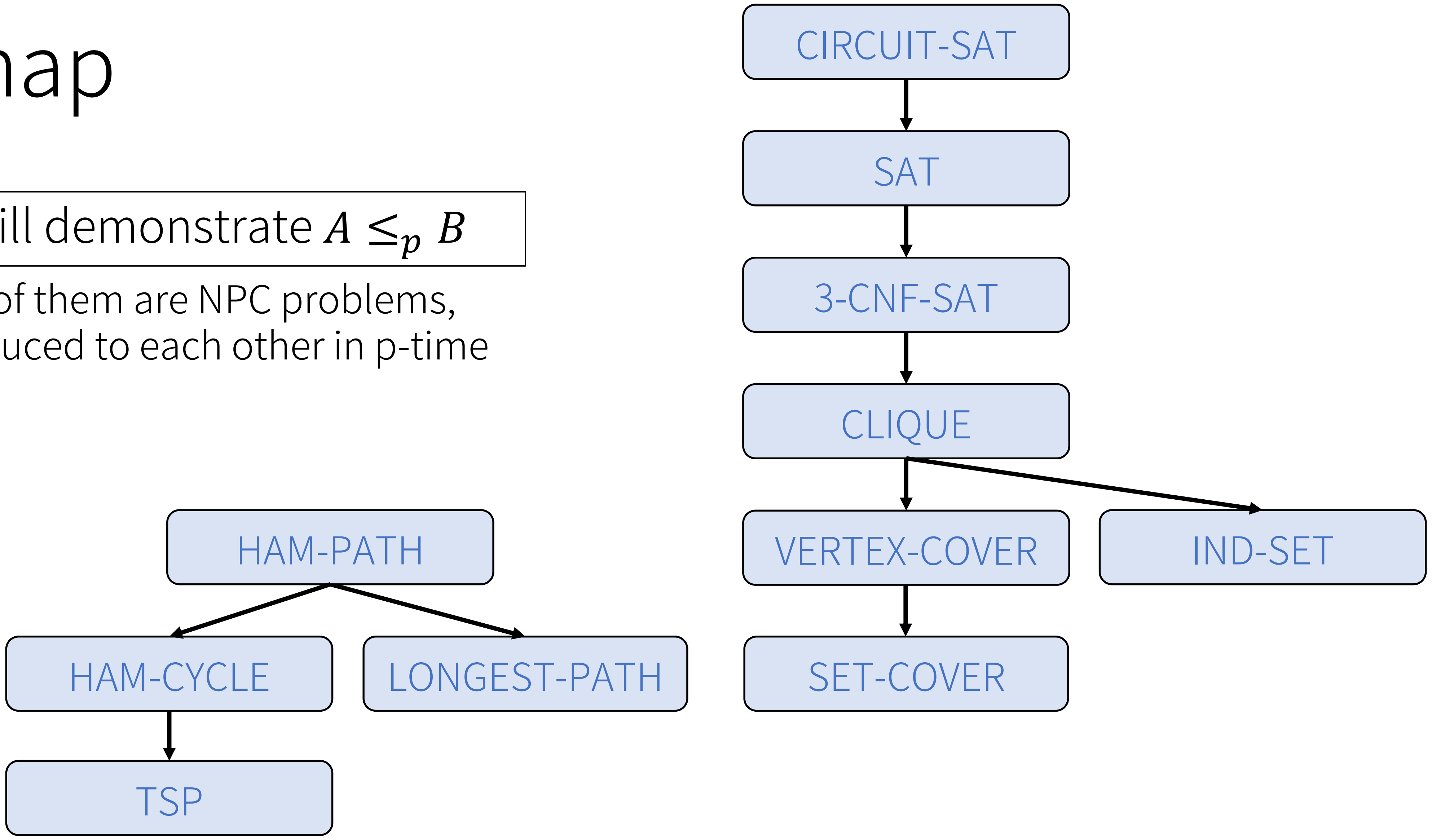
There exists a certificate for each YES instance

Classic NP-Complete Problems

Roadmap

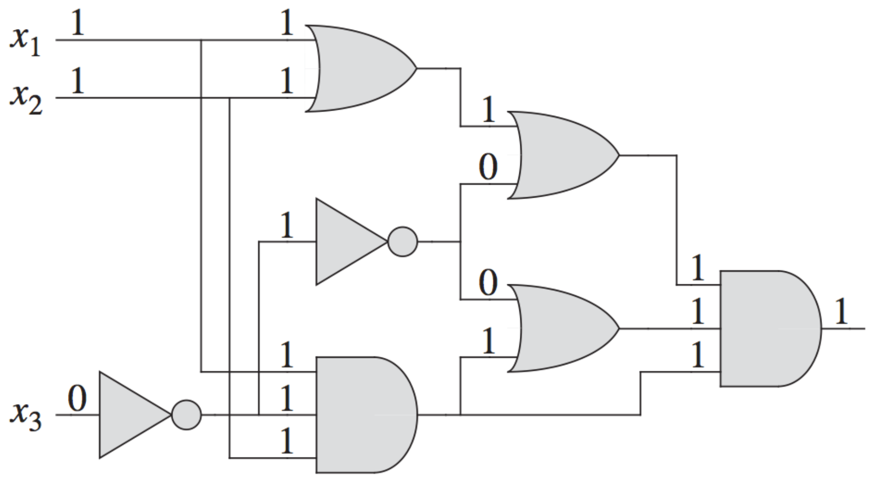
$A \rightarrow B$: we will demonstrate $A \leq_p B$

* Note that all of them are NPC problems, and can be reduced to each other in p-time



Circuit-Satisfiability Problem

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



Cook’s Theorem

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

Formula satisfiability problem (SAT)

SAT = $\{\phi | \phi$ 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: $\phi = \left((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)\right) \wedge \neg x_2$
  - $\phi$is satisfiable, and <0, 0, 1, 1> is one solution

p	q	p^q	pvq	¬p	p→q	p↔q
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

Formula satisfiability problem (SAT)

$\mathbf{SAT} = \{\phi | \phi$ is a Boolean Formula with a satisfying assignment $\}$

- Is $\mathbf{SAT} \in$ NP-Complete?
- To prove that $\mathbf{SAT}$ is NP-Complete, we show that

1. $\mathbf{SAT} \in$ NP
2. $\mathbf{SAT} \in$ NP-hard (CIRCUIT-SAT $\leq_p$ $\mathbf{SAT}$)
   ① CIRCUIT-SAT is a known NPC problem
   ② Construct a reduction f transforming every instance of CIRCUIT-SAT to an instance of $\mathbf{SAT}$
   ③ Prove that $x \in$ CIRCUIT-SAT $\Leftrightarrow f(x) \in \mathbf{SAT}$
   ④ Prove that f is a polynomial time transformation

SAT ∈ NP

- Polynomial-time verification: replaces each variable in the formula with the corresponding value in the certificate and then evaluates the expression
- Example:
  - $\phi = \left((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)\right) \wedge \neg x_2$
  - A certificate for Φ is <0, 0, 1, 1>

Textbook Ch.34-4: To show that SAT belongs to NP, we show that a certificate consisting of a satisfying assignment for an input formula ϕ can be verified in polynomial time. The verifying algorithm simply replaces each variable in the formula with its corresponding value and then evaluates the expression, much as we did in equation (34.2) above. This task is easy to do in polynomial time. If the expression evaluates to 1, then the algorithm has verified that the formula is satisfiable.

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

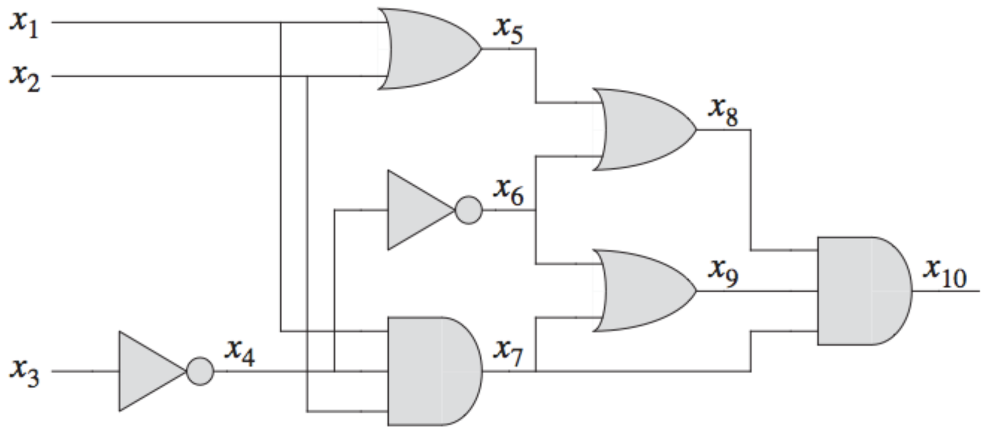
Exercises 34.4-1 Consider a straightforward reduction that simply expresses the output using the input variables only. Describe a circuit of size n that, when converted to a formula by this method, yields a formula whose size is exponential in n.

SAT ∈ NP-hard

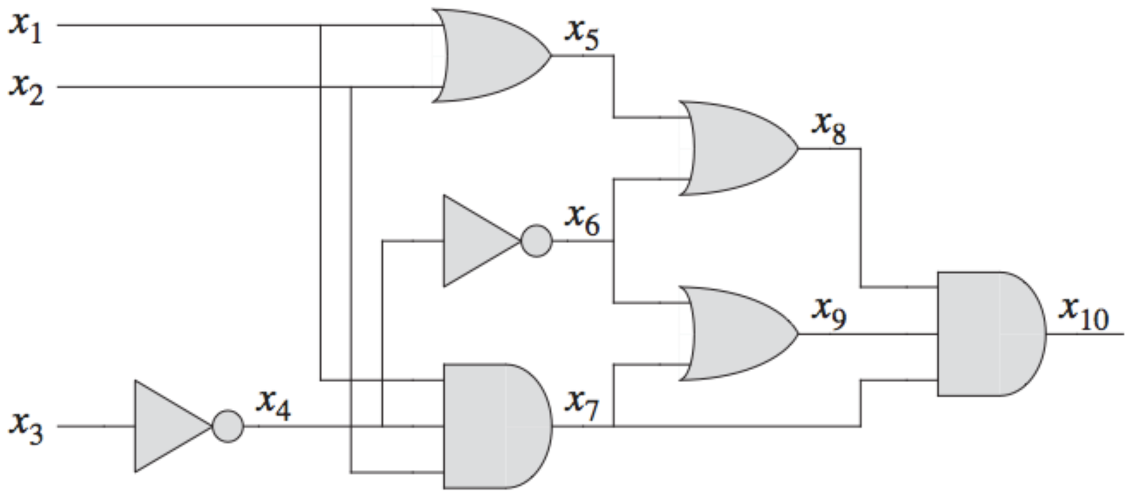
① CIRCUIT-SAT is a known NPC problem

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., $x_{10} \leftrightarrow x_7 \wedge x_8 \wedge x_9$
- ϕ = AND the output variable and the operations of all gates



SAT ∈ NP-hard

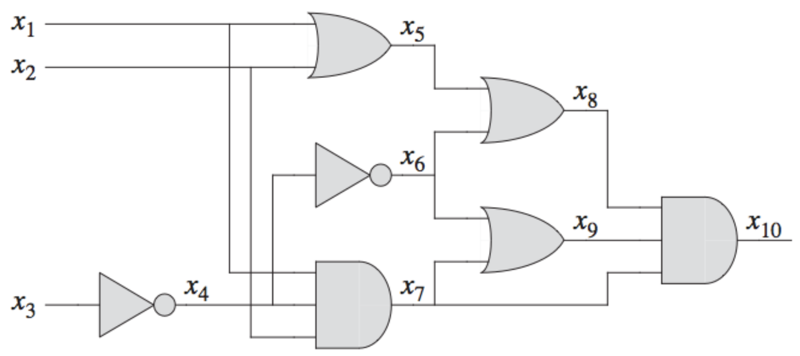


$$\begin{aligned}\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3)\\ &\wedge (x_5 \leftrightarrow (x_1 \vee x_2))\\ &\wedge (x_6 \leftrightarrow \neg x_4)\\ &\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4))\\ &\wedge (x_8 \leftrightarrow (x_5 \vee x_6))\\ &\wedge (x_9 \leftrightarrow (x_6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

SAT ∈ NP-hard

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

④ Prove that 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-SAT)

3-CNF-SAT= $\{\phi \mid \phi$ is a Boolean Formula in 3-conjunctive normal form (3-CNF) with a satisfying assignment $\}$

- Terms
  - Literal: 文字, a variable or its negation, e.g., x_1 or $\neg x_1$
  - Disjunctive : 析取, OR
  - Conjunctive: 合取, AND
  - Clause: 子句
  - Conjunctive normal form (CNF): 合取範式, conjunctive of disjunctive clauses
  - Disjunctive normal form (DNF): 析取範式, disjunctive of conjunctive clauses
  - 3-CNF: CNF where each clause has exactly 3 distinct literals
- Example: $\phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$
  - $\phi$is satisfiable, and <0, 0, 1, 1> is one solution

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

3-CNF-SAT= $\{\phi \mid \phi$ is a Boolean Formula in 3-conjunctive normal form (3-CNF) with a satisfying assignment $\}$

- Is 3-CNF-SAT ∈ NP-Complete?
- To prove 3-CNF-SAT is NP-Complete, we show that

1. 3-CNF-SAT ∈ NP
2. 3-CNF-SAT ∈ NP-hard (SAT $\leq_p$3-CNF-SAT)
   ① SAT is a known NPC problem
   ② Construct a reduction f transforming every instance of $\mathbf{SAT}$ to an instance of 3-CNF-SAT
   ③ Prove that x ∈ SAT ⇔ $f(x)$ ∈ 3-CNF-SAT
   ④ Prove that f is a polynomial time transformation

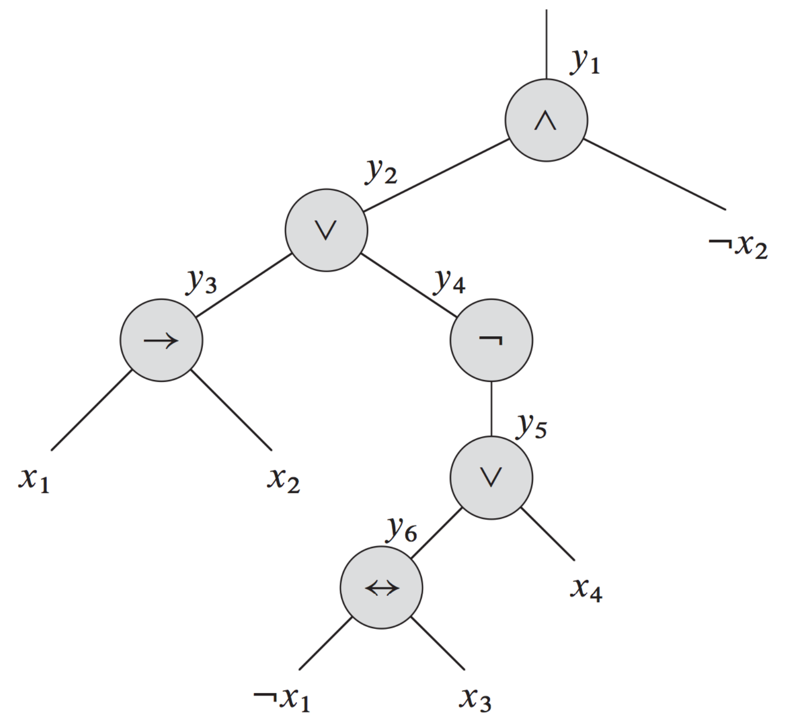
* We will focus on the construction of reduction functions from now on; but a full proof requires showing the other conditions are true as well

Practice: SAT $\leq_p$ 3-CNF-SAT

② Construct a reduction f transforming every instance of SAT to an instance of 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

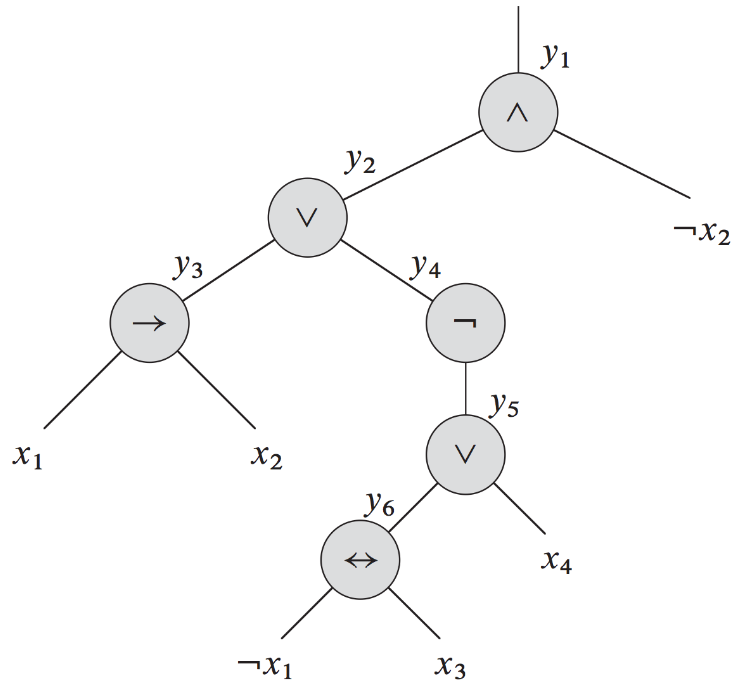
$$\phi = \left((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)\right) \wedge \neg x_2$$



Practice: SAT $\leq_p$ 3-CNF-SAT

② Construct a reduction f transforming every instance of SAT to an instance of 3-CNF-SAT

b) Construct ϕ' as the AND of the root variable and clauses describing the operation of each node



$$\begin{aligned}
\phi' = y_1 &\wedge \overset{\phi'_1}{\boxed{(y_1 \leftrightarrow (y_2 \wedge \neg x_2))}} \\
&\wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
&\wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
&\wedge (y_4 \leftrightarrow \neg y_5) \\
&\wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
&\wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\end{aligned}$$

Practice: SAT $\leq_p$ 3-CNF-SAT

② Construct a reduction f transforming every instance of SAT to an instance of 3-CNF-SAT

c) Convert each clause ϕ_i' to CNF

- Construct a truth table for each clause ϕ_i'
- Construct the DNF formula for $\neg\phi_i'$
- Apply DeMorgan's Law to $\neg\phi_i'$ and get the CNF formula ϕ_i''

DeMorgan's Law

$$\neg(a \wedge b) = \neg a \vee \neg b$$
$$\neg(a \vee b) = \neg a \wedge \neg b$$

y_1	y_2	x_2	ϕ_1' $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$	$\neg\phi_1'$
1	1	1	0	1
1	1	0	1	0
1	0	1	0	1
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	1	0
0	0	0	1	0

$$\neg\phi_1' = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

$$\phi_1'' = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

Practice: SAT $\leq_p$ 3-CNF-SAT

② Construct a reduction f transforming every instance of SAT to an instance of 3-CNF-SAT

d) Construct ϕ''' s.t. each clause C_i has exactly 3 distinct literals

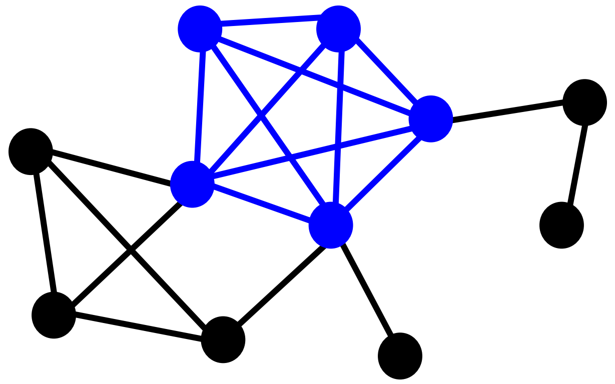
- C_i has 3 distinct literals: do nothing
- C_i has 2 distinct literals: $C_i = I_1 \vee I_2 = (I_1 \vee I_2 \vee p) \wedge (I_1 \vee I_2 \vee \neg p)$
- C_i has 1 literal: $C_i = I = (I \vee p \vee q) \wedge (I \vee \neg p \vee q) \wedge (I \vee p \vee \neg q) \wedge (I \vee \neg p \vee \neg q)$

③ Prove that $x \in \mathrm{SAT} \iff f(x) \in$ 3-CNF-SAT

- Show that ϕ''' is satisfiable iff ϕ is satisfiable

The clique problem

- A clique in an undirected $G = (V, E)$ is a subset of $V' \subseteq V$ in which each pair of vertices are connected by an edge in E
- 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

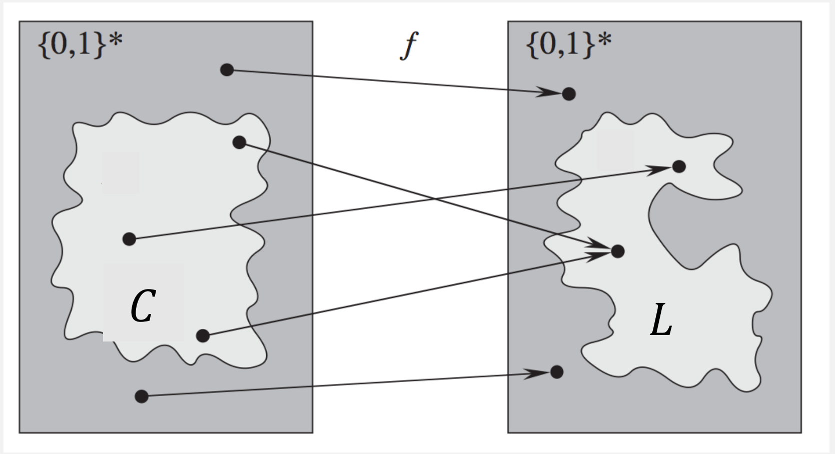
The Clique Problem

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

- Prove that CLIQUE ∈ NP-COMPLETE
- Polynomial-time reduction : 3-CNF-SAT $\leq_p$ CLIQUE

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

1. Prove $L \in$ NP
2. Prove $L \in$ 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



The Clique Problem

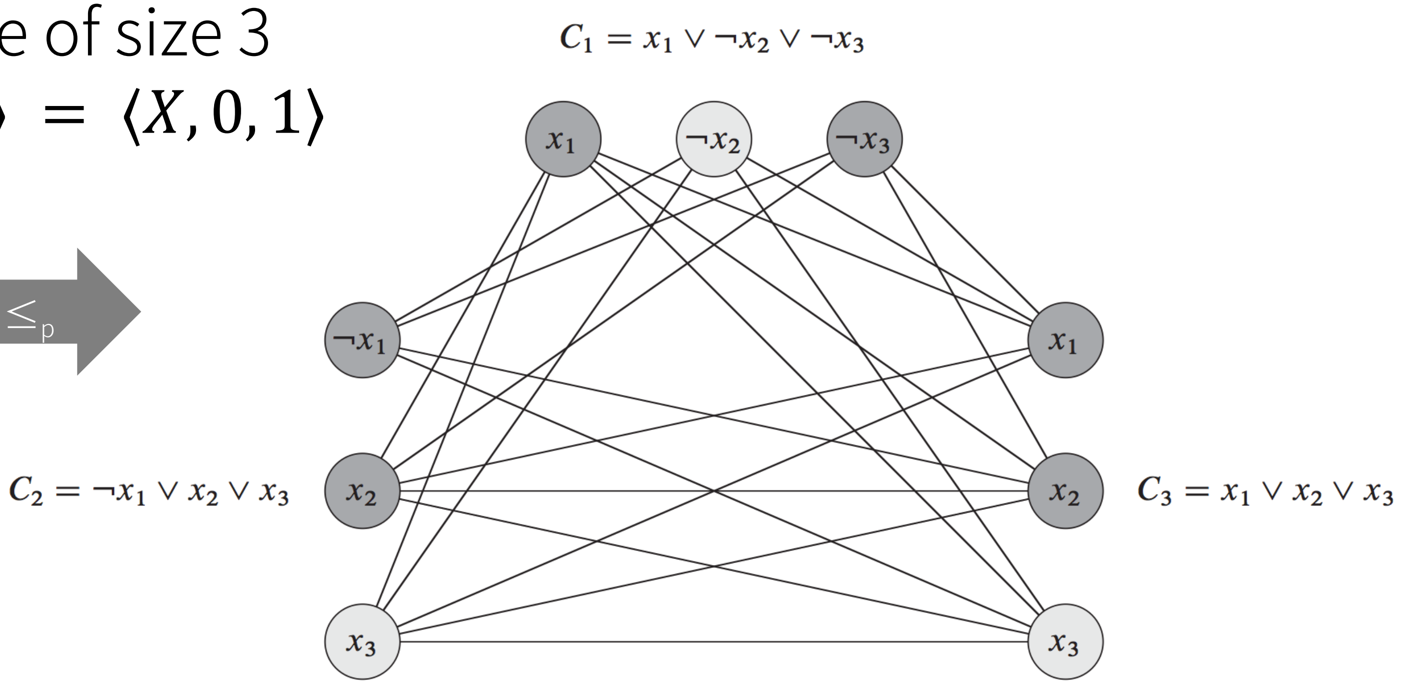
CLIQUE = $\{\langle G, k\rangle : G$ is a graph containing a clique of size $k\}$

- Prove that CLIQUE ∈ NP-COMPLETE
- Polynomial-time reduction : 3-CNF-SAT $\leq_p$ CLIQUE
  - Construct a graph G s.t. ϕ with k clauses is satisfiable ⇔ $G = f(\phi)$ has a clique of size k

Example: $\phi = 1 \Leftrightarrow G$ has a clique of size 3
Satisfying assignment $\langle x_1, x_2, x_3\rangle = \langle X, 0, 1\rangle$

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

$\xrightarrow{\leq_p}$



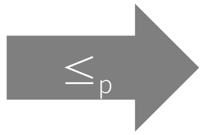
$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
$C_2 = \neg x_1 \vee x_2 \vee x_3$
$C_3 = x_1 \vee x_2 \vee x_3$
x_1
$\neg x_2$
$\neg x_3$
$\neg x_1$
x_2
x_3
x_1
x_2
x_3

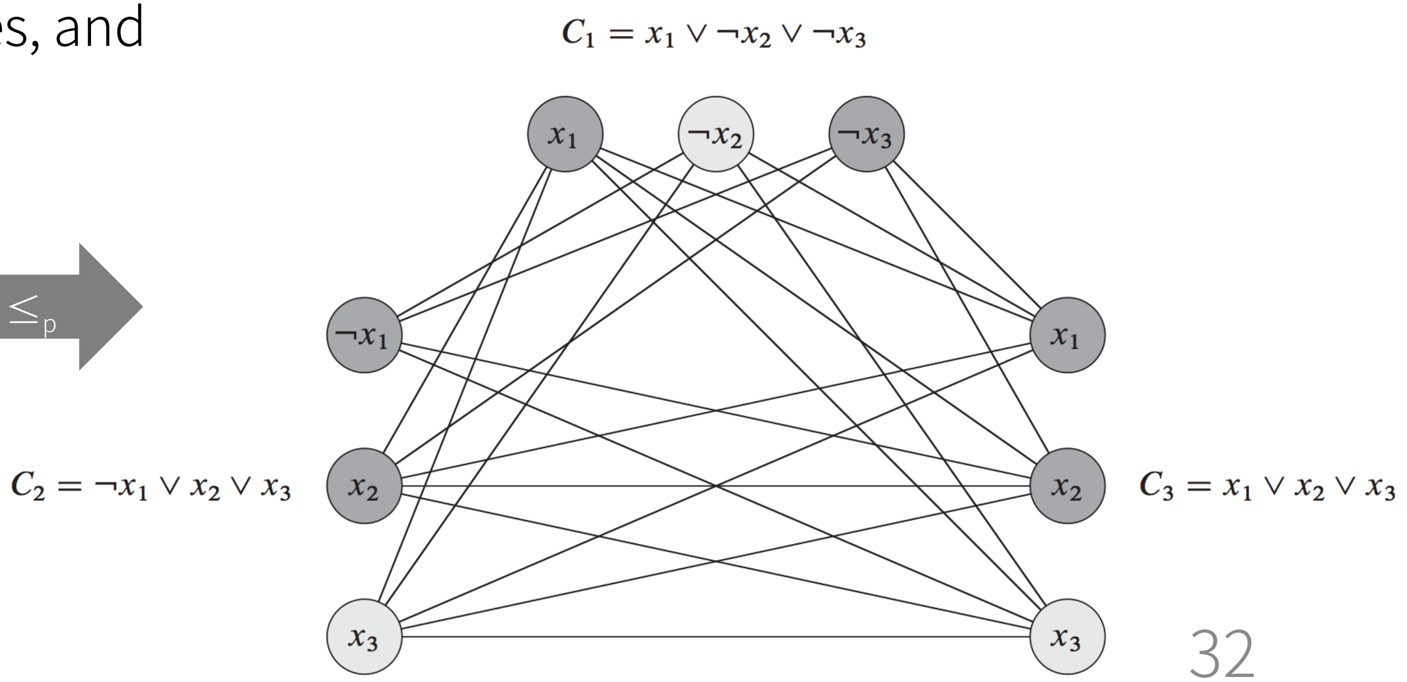
3-CNF-SAT $\leq_p$ CLIQUE

② Construct a reduction f transforming every instance of 3-CNF-SAT to an instance of CLIQUE

- Let $\phi = C_1 \land C_2 \land \cdots \land C_k$ 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 $C_r = (l_1^r \lor l_2^r \lor l_3^r)$ 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

$$\phi = (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3)$$




$C_1 = x_1 \lor \neg x_2 \lor \neg x_3$
x_1
$\neg x_2$
$\neg x_3$
$C_2 = \neg x_1 \lor x_2 \lor x_3$
$\neg x_1$
x_2
x_3
$C_3 = x_1 \lor x_2 \lor x_3$
x_1
x_2
x_3

3-CNF-SAT $\leq_p$ CLIQUE

③ Prove that $x \in$ 3-CNF-SAT $\Leftrightarrow$ $f(x) \in$ CLIQUE

<u>Correctness proof</u>: ϕ with k clauses is satisfiable $\Rightarrow$ $G = f(\phi)$ has a clique of size k

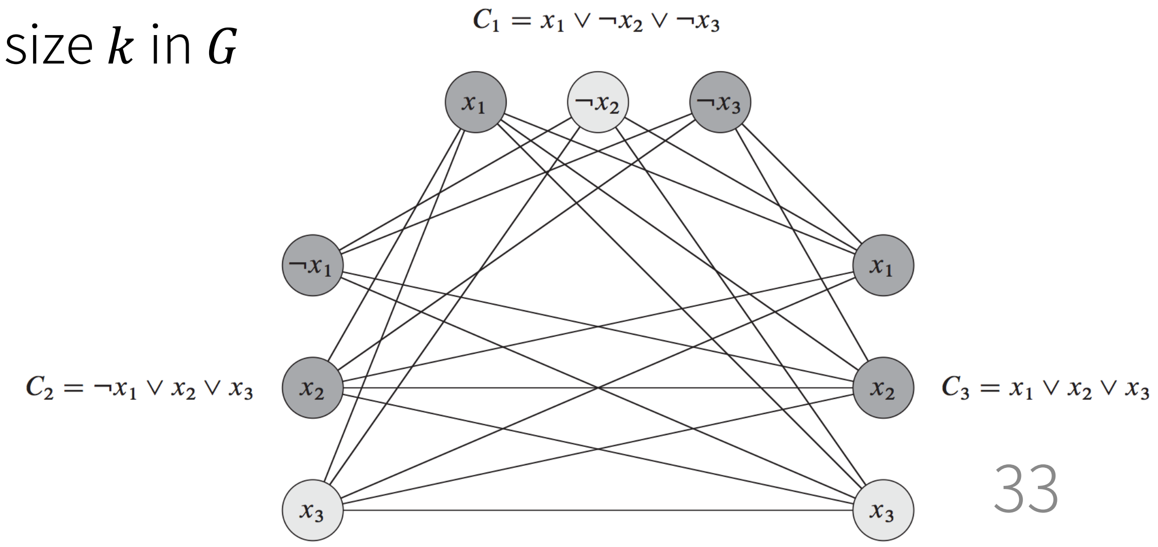
If ϕ is satisfiable,

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

We select such a literal from each C_r and form a set k literals

$\Rightarrow$ For any two selected literals $l_i^r, l_j^s, r \neq s$ and $l_i^r \neq \neg l_j^s$

$\Rightarrow$ The corresponding vertices form a clique of size k in G


$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
x_1
$\neg x_2$
$\neg x_3$
$\neg x_1$
x_2
x_3
$C_2 = \neg x_1 \vee x_2 \vee x_3$
x_1
x_2
x_3
$C_3 = x_1 \vee x_2 \vee x_3$

3-CNF-SAT $\leq_p$ CLIQUE

③ Prove that $x \in$ 3-CNF-SAT $\Leftrightarrow f(x) \in$ CLIQUE

<u>Correctness proof (cont'd):</u> $G = f(\phi)$ has a clique of size $k \Rightarrow \phi$ with k clauses is satisfiable

G has a clique V' of size k

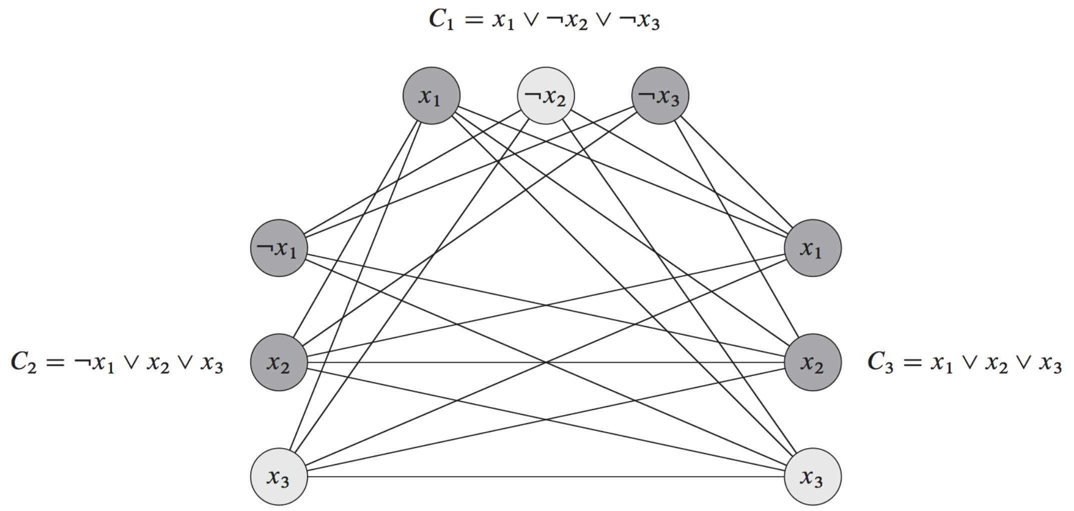
$\Rightarrow$ For all $v_i^r, v_j^s \in V'$, $(v_i^r, v_j^s) \in E$

$\Rightarrow$ For all $v_i^r, v_j^s \in V'$, $l_i^r \neq \neg l_j^s$

$\Rightarrow$ Assigning 1 to each l_i^r where $v_i^r \in V'$ results in a valid assignment

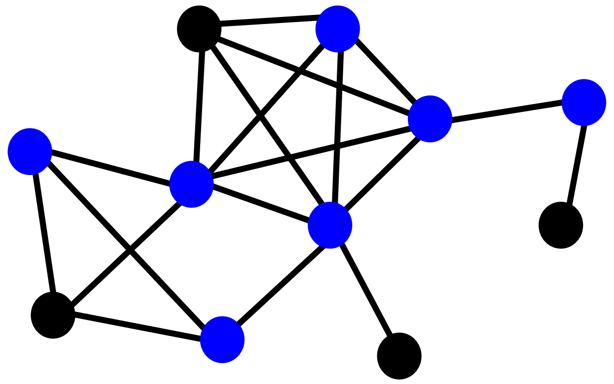
Also, V' must contain exactly one vertex per triple since no edges are within the same triple

$\Rightarrow$ Each C_r is satisfied, and so is ϕ


$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
x_1
$\neg x_2$
$\neg x_3$
$C_2 = \neg x_1 \vee x_2 \vee x_3$
$\neg x_1$
x_2
x_3
$C_3 = x_1 \vee x_2 \vee x_3$
x_1
x_2
x_3

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
- <u>Optimization problem</u>: find a vertex cover of minimum size in G
- <u>Decision problem</u>: 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?

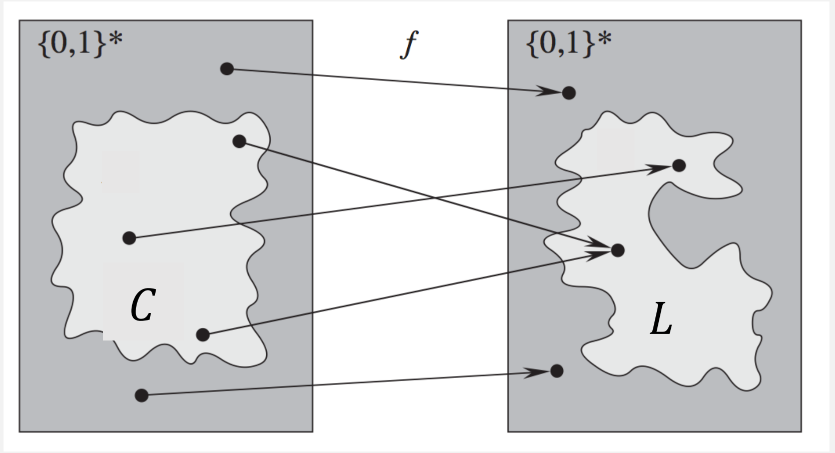
The Vertex-Cover Problem

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

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

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

1. Prove $L \in$ NP
2. Prove $L \in$ 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



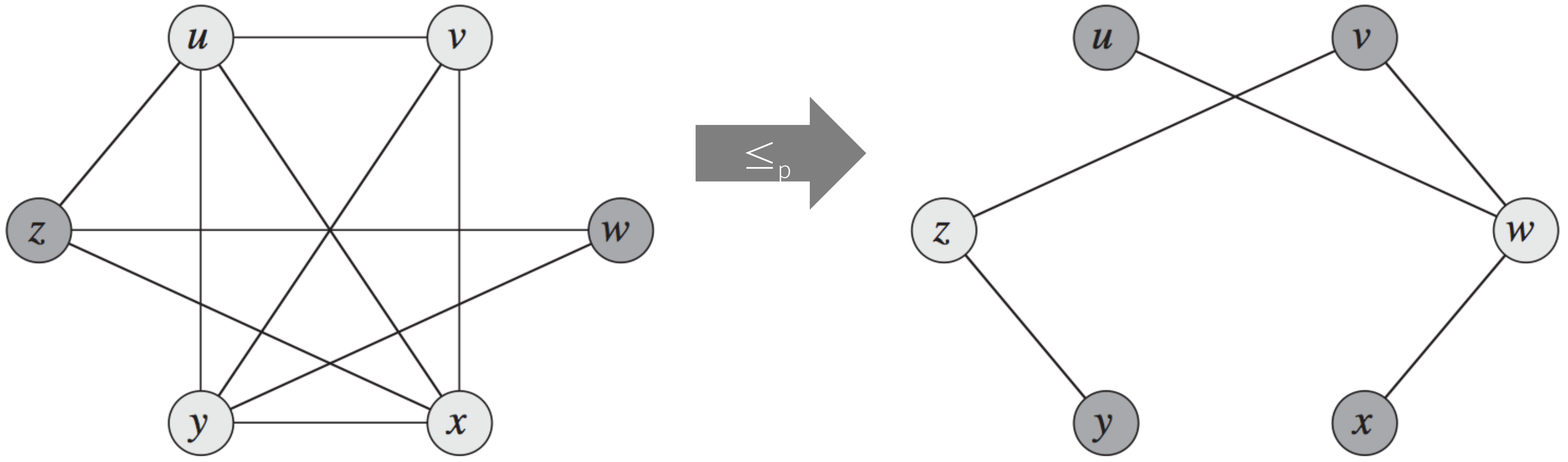
The Vertex-Cover Problem

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

② Construct a reduction f transforming every CLIQUE's instance to a VERTEX-COVER instance

- $f(\{G, k\}) = \{G_c, |V| - k\}$
- G_c is 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



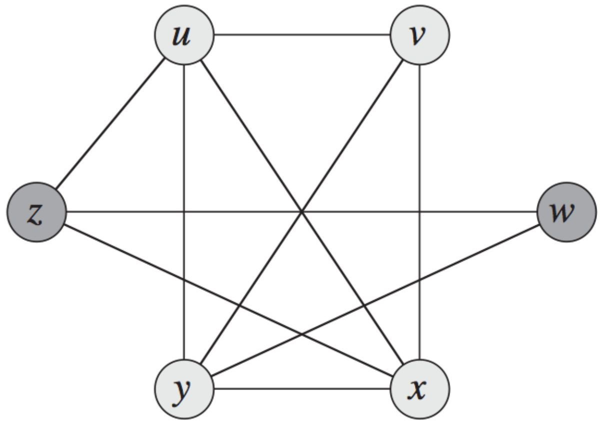
(a) (b)

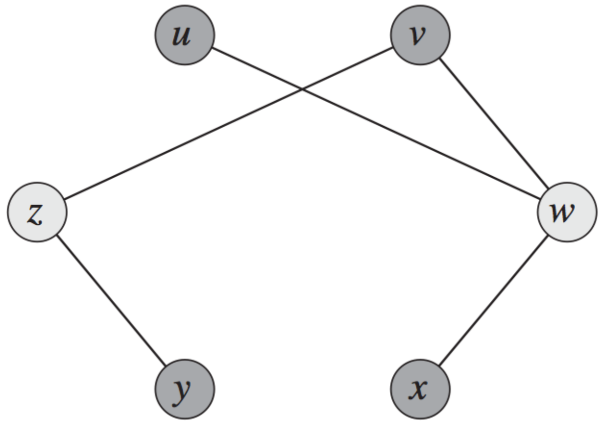
CLIQUE $\leq_p$ VERTEX-COVER

③ Prove that x ∈ CLIQUE ⇔ $f(x)$ ∈ VERTEX-COVER

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

- Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$
- $\Rightarrow$ for all $(w, v) \notin E$ (i.e, $\in E_c$), 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) in G_c is covered by $V - V'$
- $\Rightarrow V - V'$ forms a vertex cover of G_c, and $|V - V'| = |V| - k$



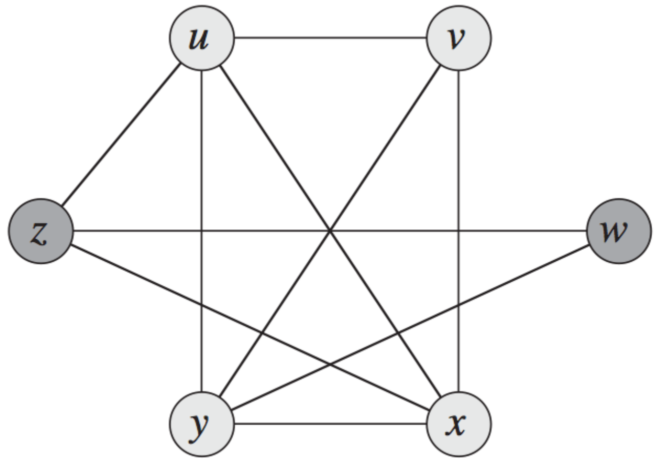


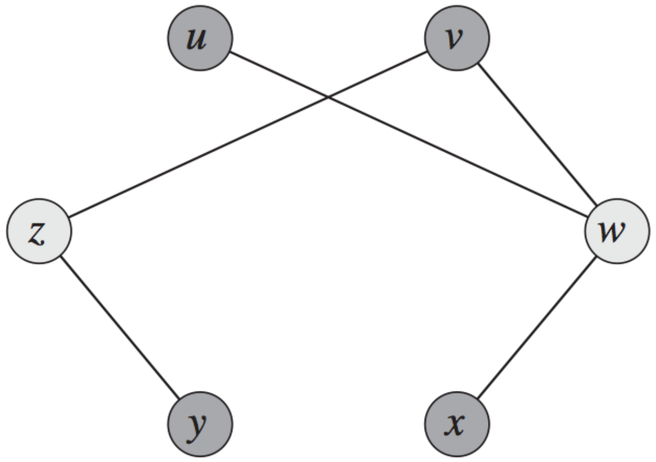
CLIQUE $\leq_p$ VERTEX-COVER

③ Prove that $x \in$ CLIQUE $\Leftrightarrow f(x) \in$ VERTEX-COVER

<u>Correctness proof (cont'd):</u> G_c has as a vertex cover of size $|V| - k \Rightarrow G$ has a clique of size k

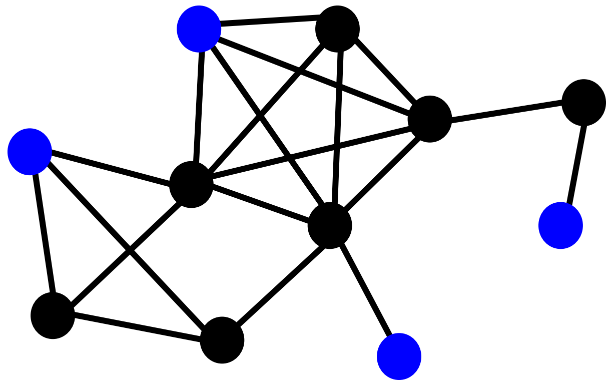
- Suppose G_c has a vertex cover $V' \subseteq V$ with $|V'| = |V| - k$
- $\Rightarrow \forall\, u, v \in V$, if $(u, v) \in E_c$, then $u \in V'$ or $v \in V'$ or both
- $\Rightarrow \forall\, u, v \in V$, if $u \notin V'$ and $v \notin V'$, then $(u, v) \notin E_c$; that is, $(u, v) \in E$
- $\Rightarrow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

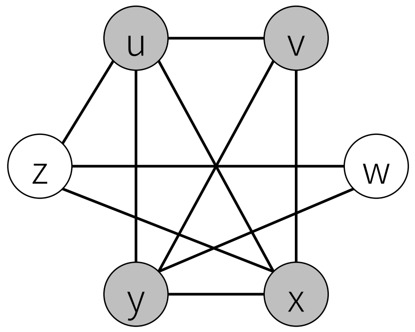
Does G have an independent set of size 4?

Does G have an independent set of size 5?

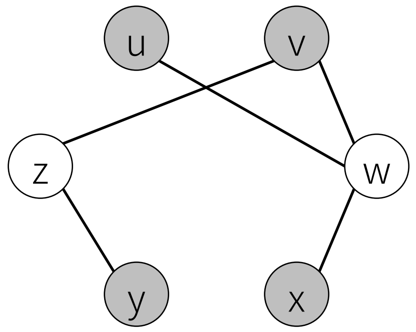
Practice: show that IND-SET is NP-Complete (Textbook Problem 34-1)

Clique, Independent-Set, Vertex-Cover

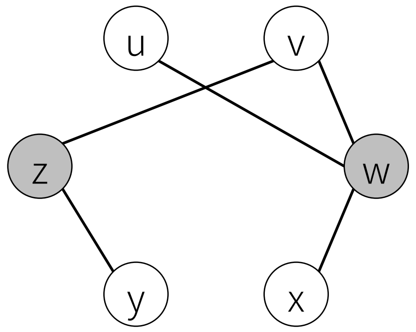
- The following are equivalent for $G = (V, E)$ and a subset V' of V:
  1. V' is a clique of G
  2. V' is an independent set of G_c
  3. $V - V'$ is a vertex cover of G_c



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

The set-cover problem

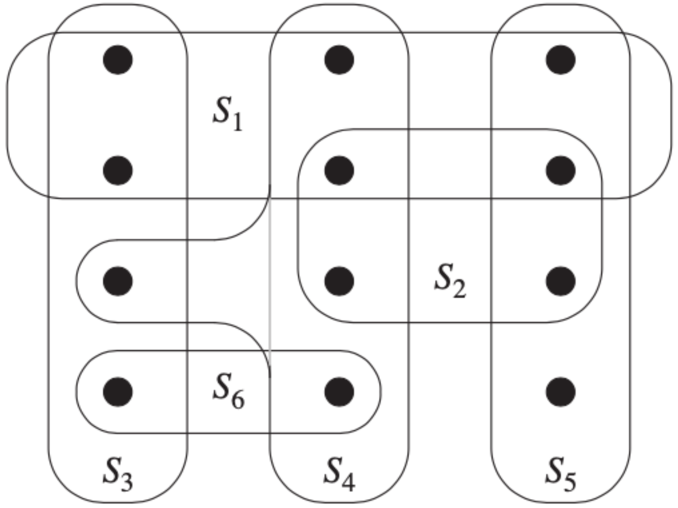
An instance $(X, \mathcal{F})$ of the ***set-covering problem*** consists of a finite set X and a family $\mathcal{F}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S .$$

We say that a subset $S \in \mathcal{F}$ ***covers*** its elements. The problem is to find a minimum-size subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X:

$$X = \bigcup_{S \in \mathcal{C}} S . \tag{35.8}$$

We say that any $\mathcal{C}$ satisfying equation (35.8) ***covers*** X. Figure 35.3 illustrates the set-covering problem. The size of $\mathcal{C}$ is the number of sets it contains, rather than the number of individual elements in these sets, since every subset $\mathcal{C}$ that covers X must contain all $|X|$ individual elements. In Figure 35.3, the minimum set cover has size 3.



Practice: show that SET-COVER is NP-Complete (Textbook Exercise 35.3-2)

Handling NP-completeness

To solve or cope with it

Design an algorithm to solve the computational problem efficiently

Prove that the problem is too hard to solve (no easier than solving NP-complete problems!)

… and then
try to cope with it!

Coping with NP-hard problems

Since polynomial-time solutions are unlikely (unless P = NP), we must sacrifice either optimality, efficiency, or generality

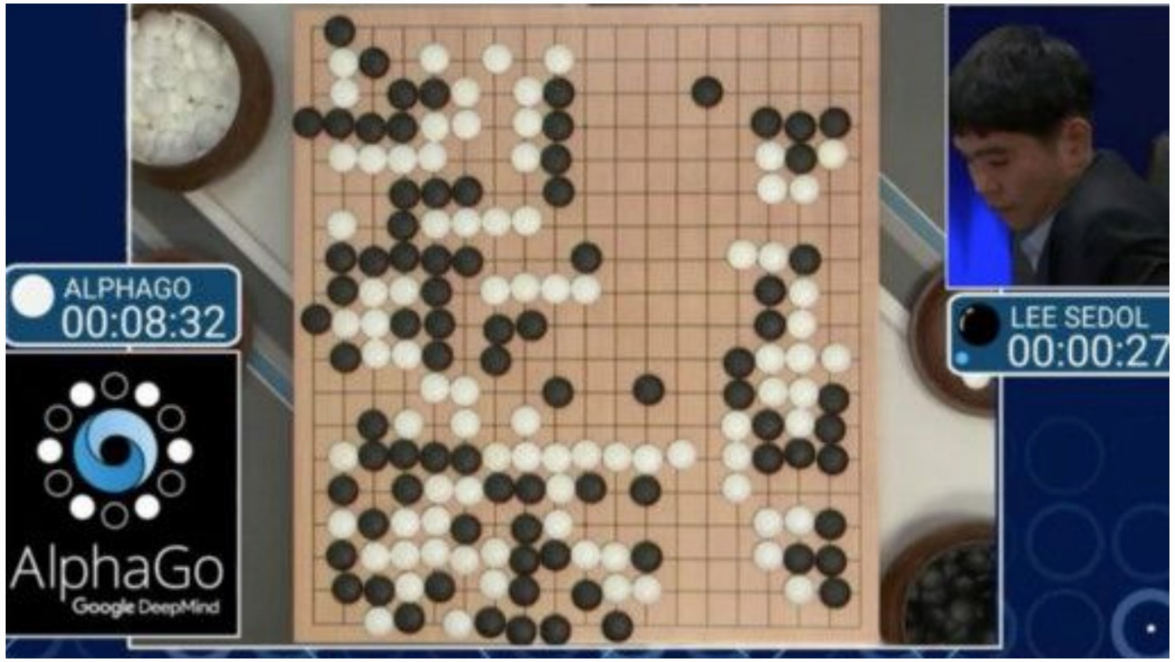
- Approximation algorithms: guarantee to be a fixed ratio away from the optimum
- Randomized algorithms: make use of a randomizer for operation
- Local search: simulated annealing (hill climbing), genetic algorithms, etc.
- Exponential algorithms/Intelligent exhaustive search: feasible when problem size is small
- Pseudo-polynomial time algorithms: e.g., DP for the knapsack problem
- Restriction: work on some special cases of the original problem. e.g., the maximum independent set problem in circle graphs

BBC NEWS | 中文

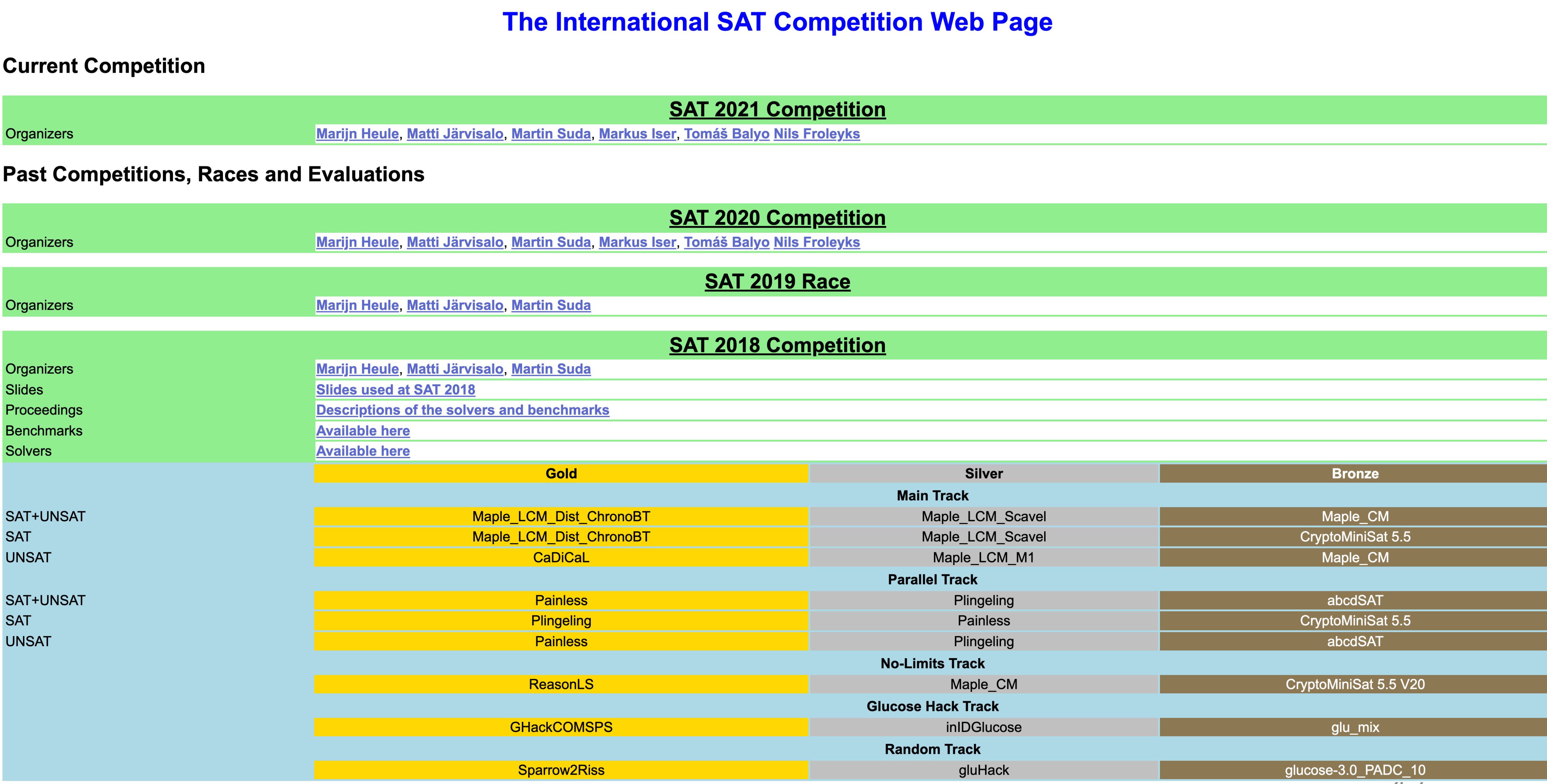
主頁 國際 兩岸 英國 科技 財經 視頻材料 BBC英倫網

人工智能AlphaGo三局完勝圍棋冠軍李世石

2016年3月12日



谷歌人工智能系統**AlphaGo**周六（**3**月**12**日）在韓國首都首爾第三次擊敗世界圍棋冠軍李世石。



The International SAT Competition Web Page

Current Competition

SAT 2021 Competition	
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda, Markus Iser, Tomáš Balyo Nils Froleyks

Past Competitions, Races and Evaluations

SAT 2020 Competition	
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda, Markus Iser, Tomáš Balyo Nils Froleyks

SAT 2019 Race	
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda

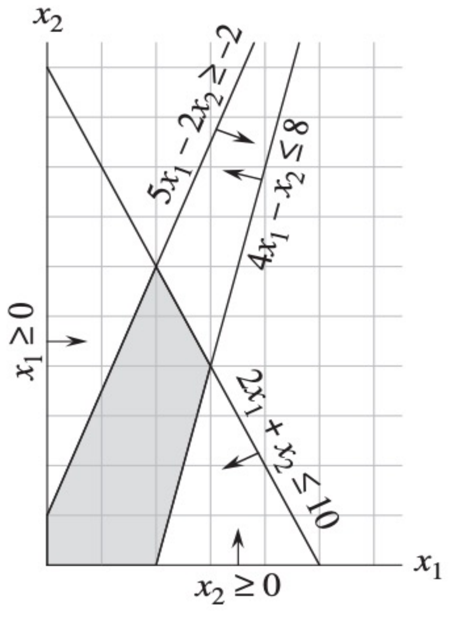
SAT 2018 Competition			
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda		
Slides	Slides used at SAT 2018		
Proceedings	Descriptions of the solvers and benchmarks		
Benchmarks	Available here		
Solvers	Available here		
	Gold	**Silver**	**Bronze**
		Main Track	
SAT+UNSAT	Maple_LCM_Dist_ChronoBT	Maple_LCM_Scavel	Maple_CM
SAT	Maple_LCM_Dist_ChronoBT	Maple_LCM_Scavel	CryptoMiniSat 5.5
UNSAT	CaDiCaL	Maple_LCM_M1	Maple_CM
		Parallel Track	
SAT+UNSAT	Painless	Plingeling	abcdSAT
SAT	Plingeling	Painless	CryptoMiniSat 5.5
UNSAT	Painless	Plingeling	abcdSAT
		No-Limits Track	
	ReasonLS	Maple_CM	CryptoMiniSat 5.5 V20
		Glucose Hack Track	
	GHackCOMSPS	inIDGlucose	glu_mix
		Random Track	
	Sparrow2Riss	gluHack	glucose-3.0_PADC_10

http://www.satcompetition.org/

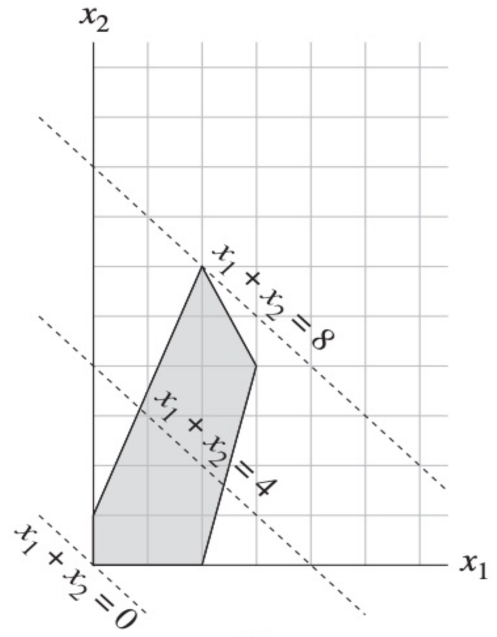
Handling NP-completeness: Integer Linear Programming and Solvers

Linear programming [Ch. 29]

- Optimize a linear function subject to a set of linear inequalities
- Example
  - Maximize $x_1 + x_2$
  - Subject to
    - $4x_1 - x_2 \leq 8$
    - $2x_1 + x_2 \leq 10$
    - $5x_1 - 2x_2 \geq -2$
    - $x_1, x_2 \geq 0$



(a)



(b)

Linear programming [Ch. 29]

Maximize $c^T x$

Subject to $Ax \leq b$, and $x \geq 0$

Notation

- $A = (a_{ij})$: $m \times n$ coefficient matrix
- $b = (b_i)$: $\mathrm{m} \times 1$ requirement vector
- $c = (c_j)$: $n \times 1$ cost vector
- $x = (x_j)$: $n \times 1$ vector of unknown variables

Integer (linear) programming [ch. 35.4]

- Integer programing: x_j are integers
    - Decision problem: whether there is a feasible solution $x = (x_j)$ subject to $Ax \leq b$ and $x_i \in \mathbb{Z}_0^+$
- Mixed integer programming: some of x_j are integers
- While the linear programing problem is in class P, the integer programming problem (decision version) is NP-complete
- Fortunately, there are powerful integer programming solvers, which haven solved many integer programming instances

Show that the integer programming problem is NP-complete

Hint: Consider a reduction from 3-CNF-SAT. How would you reduce an 3-CNF-SAT instance, say, $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_4 \vee x_5)$, to an integer programming instance?

Modeling vertex cover via integer programming

- Vertex cover (optimization): find a vertex cover of minimum size in G
  - 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
- Integer programming formulation
  - Variables: $x_i \in \{0,1\}$ represents whether vertex v_i is covered
  - Minimize: $\Sigma_{i=1}^{n} x_i$
  - Subject to
    - $x_i + x_j \geq 1, \forall e = (v_i, v_j) \in E$
    - $x_i \in \{0,1\}, \forall i = 1,2, \ldots, |V|$

Show the integer programming formulation for the clique problem (optimization)

Show the integer programming formulation for the independent-set problem (optimization)

Show the integer programming formulation for the set-cover problem (optimization)

Preview: approximation algorithms

What is approximation?

- “A value or quantity that is nearly but not exactly correct”
- Approximation algorithms
  - applied to optimization problem, not decision problems
  - should guaranteed to find solutions close to optimality, returning near-optimal answers
    - Cf. heuristics search: no guarantee
    - How “near” is near-optimal?

Approximation algorithms

- $\rho(n)$ -approximation algorithm
  - Efficient: guaranteed to run in polynomial time
  - General: guaranteed to solve every instance of the problem
  - Near-optimal: guaranteed to find solution within a factor of $\rho(n)$ of the cost of an optimal solution
- Approximation ratio $\rho(n)$
  - n: input size
  - C^*: cost of an optimal solution
  - C: cost of the solution produced by the approximation algorithm

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

Maximization problem: $\frac{C^*}{C} \le \rho(n)$

Minimization problem: $\frac{C}{C^*} \le \rho(n)$

Approximate ratio ρ(n)

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

n: input size
C^*: cost of optimal solution
C: cost of approximate solution

- $\rho(n) \geq 1$
- $\rho(n)$ 越小越好！
- An exact algorithm has $\rho(n) = 1$

- Challenge: prove that C is close to C^* without knowing C^*!

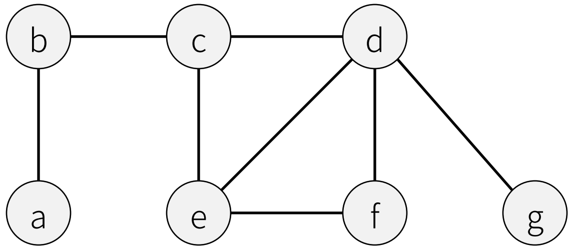
Approximate Vertex-Cover

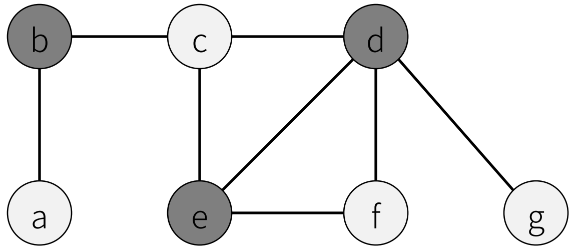
Vertex Cover

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

The Vertex-Cover Problem (Optimization)

Find a vertex cover of minimum size in G

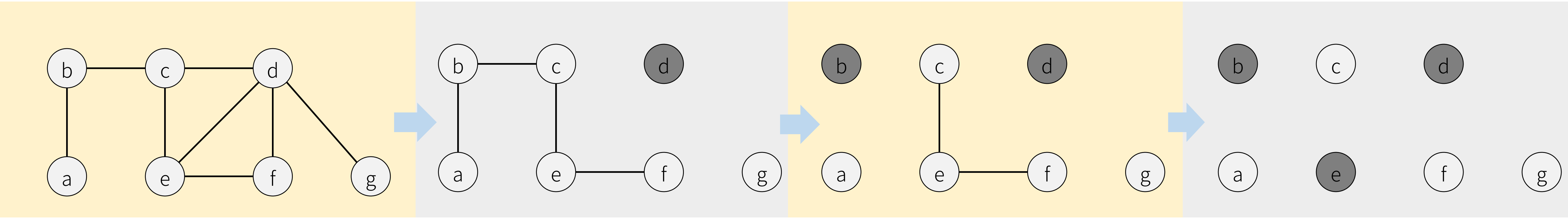




b, d, e is a minimum vertex cover
(size = 3)

Q: Consider a greedy heuristic: In each iteration, cover as many edges as possible (vertex with the maximum degree) and then delete the covered edges. Does it always find an optimal solution?

- No. Otherwise, we would have proven P=NP.



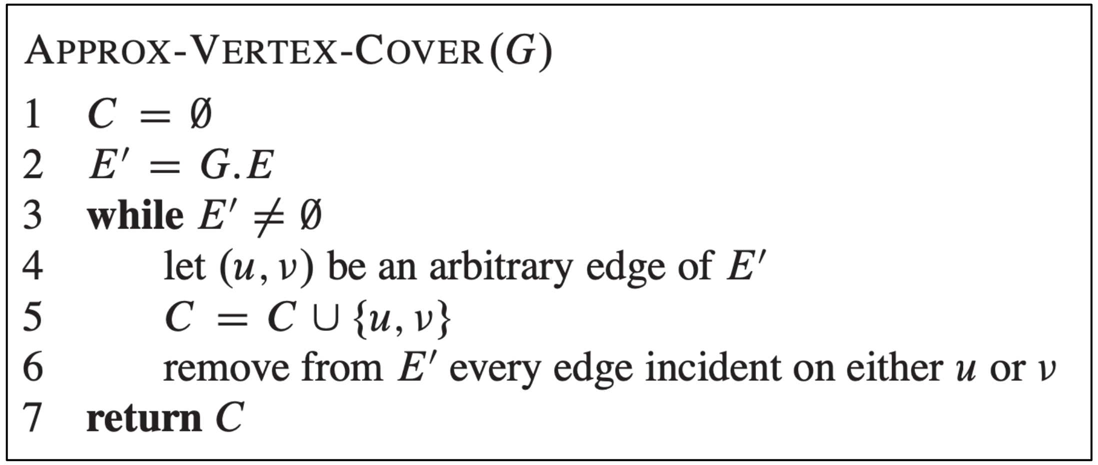
b, d, e is a vertex cover of size 3 found by the greedy algorithm (and it happens to be optimal!)

Q: Construct a graph on which the above greedy heuristic does not yield an optimal solution

Exercise 35.1-3 Construct a graph on which the above greedy heuristic does not have an approximation ratio of 2.

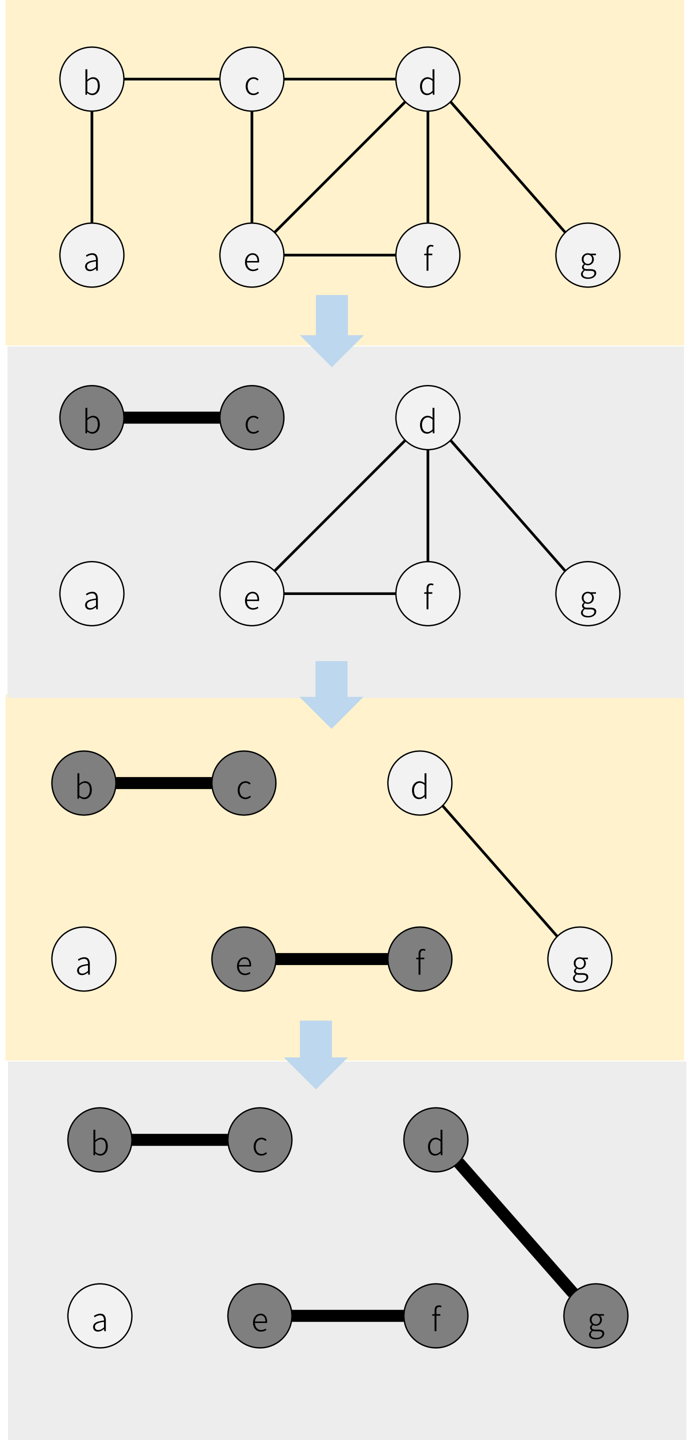
An approximation algorithm to VERTEX-COVER

- APPROX-VERTEX-COVER
  - Randomly select one edge at a time
  - Add both vertices to the cover
  - Remove all incident edges
- Running time = $O(V + E)$
- <u>Claim</u>: Approximation ratio $\rho(n) = 2$



```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
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
```

An approximation algorithm to VERTEX-COVER: example



b, c, d, e, f, g is a vertex cover of size 6 found by the approximation algorithm (not optimal!)

APPROX-VERTEX-COVER is 2-approximation

Let A denote the set of edges picked in line 4.
$\Rightarrow$ size of the vertex cover $|S| = 2|A|$

In any vertex cover S' (including S^*), every vertex v in S' covers at most one edge in A because no two edges in A share a vertex.

$\Rightarrow |A| \leq |S^*|$

Combing the two equations, we have
$\frac{1}{2}|S| = |A| \leq |S^*|$

$\Rightarrow \rho(n) = 2$

Note: the proof doesn't require knowing the actual value of $|S^*|$