NP Completeness

Classes “P” and “NP”

- **Class P** consists of decision problems that are solvable in polynomial time:
  - $O(n^k)$, k constant
- **Class NP** consists of problems that are verifiable in polynomial time
  - Could be solved by nondeterministic polynomial algorithms

Nondeterministic Algorithms

Nondeterministic algorithm = two stage procedure:

1) Nondeterministic (“guessing”) stage:

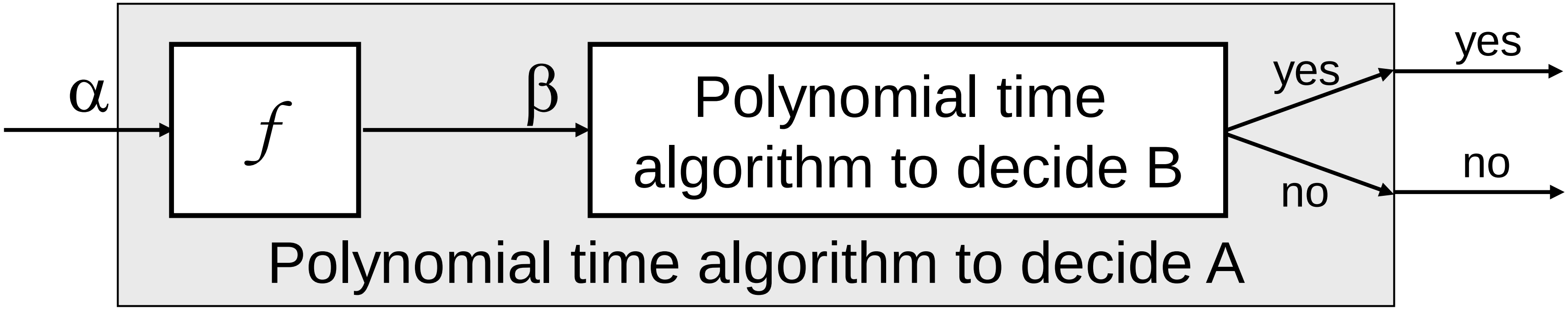
   generate an arbitrary string that can be thought of as a candidate solution (“certificate”)

2) Deterministic (“verification”) stage:

   take the certificate and the instance to the problem and returns YES if the certificate represents a solution

- **Nondeterministic polynomial** (NP) = verification stage is polynomial

Polynomial Reduction Algorithm



- To solve a decision problem A in polynomial time
  1. Use a polynomial time reduction algorithm to transform A into B
  2. Run a known polynomial time algorithm for B
  3. Use the answer for B as the answer for A

Reductions

- Given two problems A, B, we say that A is **reducible** to B ($A \leq_p B$) if:

1. There exists a function f that converts the input of A to inputs of B in polynomial time
2. $A(i) = YES \Leftrightarrow B(f(i)) = YES$

NP-Completeness

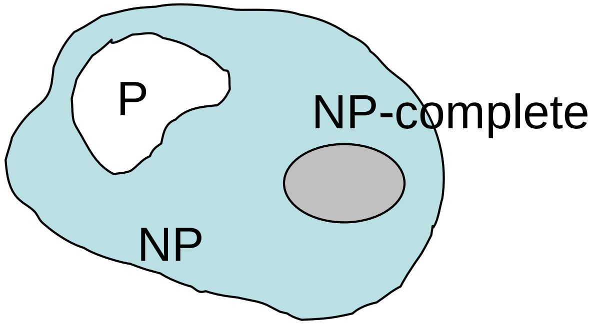
- A problem B is **NP-complete** if:

  1) B $\in$ **NP**

  2) A $\leq_p$ B for all A $\in$ **NP**

- If B satisfies only property 2) we say that B is **NP-hard**
- No polynomial time algorithm has been discovered for an **NP-Complete** problem
- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem

Is P = NP?



- Any problem in P is also in NP:

$$P \subseteq NP$$

- We can solve problems in P, even without having a certificate
- The big (and open question) is whether P = NP

Theorem: If any NP-Complete problem can be solved in polynomial time $\Rightarrow$ then P = NP.

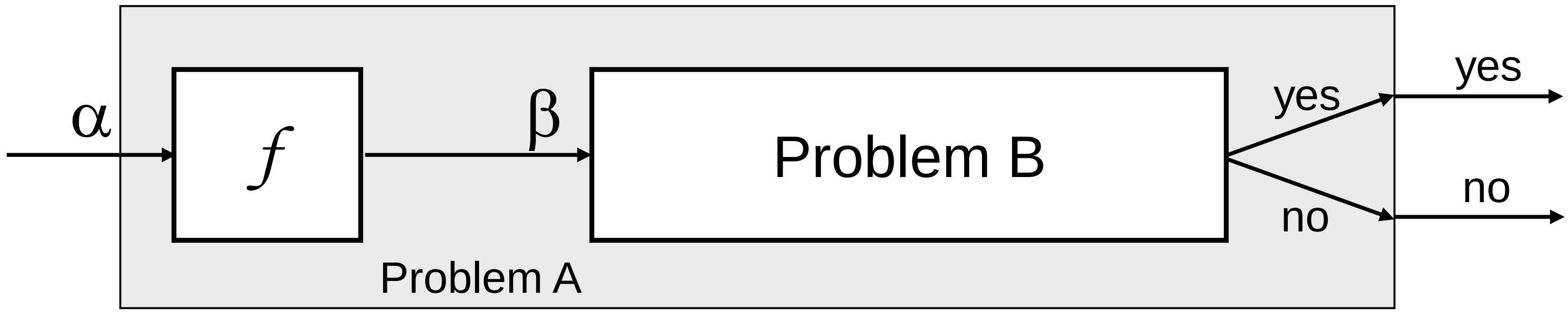
P & NP-Complete Problems

- **Shortest simple path**
  - Given a graph G = (V, E) find a **shortest** path from a source to all other vertices
  - Polynomial solution: O(VE)
- **Longest simple path**
  - Given a graph G = (V, E) find a **longest** path from a source to all other vertices
  - NP-complete

P & NP-Complete Problems

- **Euler tour**
  - G = (V, E) a connected, directed graph find a cycle that traverses each edge of G exactly once (may visit a vertex multiple times)
  - Polynomial solution O(E)
- **Hamiltonian cycle**
  - G = (V, E) a connected, directed graph find a cycle that visits each vertex of G exactly once
  - NP-complete

Reduction and NP-Completeness



- Suppose we know:
  - No polynomial time algorithm exists for problem A
  - We have a polynomial reduction f from A to B

$\Rightarrow$ No polynomial time algorithm exists for B

Proving NP-Completeness

Theorem: If A is NP-Complete and A $\leq_p$ B

$\Rightarrow$ B is NP-Hard

In addition, if B $\in$ NP

$\Rightarrow$ B is NP-Complete

Proof: Assume that B $\in$ P

Since A $\leq_p$ B $\Rightarrow$ A $\in$ P contradiction!

$\Rightarrow$ B is NP-Hard

Proving NP-Completeness

- Prove that the problem B is in NP
  - A randomly generated string can be checked in polynomial time to determine if it represents a solution
- Show that **one known** NP-Complete problem can be transformed to B in polynomial time
  - No need to check that **all** NP-Complete problems are reducible to B

Boolean Formula Satisfiability

Formula Satisfiability Problem: a boolean formula Φ composed of

1. n boolean variables: $x_1, x_2, \ldots, x_n$
2. m boolean connectives: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\leftrightarrow$ if and only if
3. Parenthesis

Satisfying assignment: an assignment of values (0, 1) to variables x_i that causes Φ to evaluate to 1

E.g.: $\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$

Certificate: $x_1 = 1, x_2 = 0 \Rightarrow \Phi = 1 \wedge 1 \wedge 1 = 1$

– Formula Satisfiability is first to be proven NP-Complete

3-CNF Satisfiability

3-CNF Satisfiability Problem:

- n boolean variables: $x_1, x_2, \ldots, x_n$
- **Literal**: x_i or $\neg x_i$ (a variable or its negation)
- **Clause:** c_j = an **OR** of **three literals** (m clauses)
- Formula: $\Phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$

- *E.g.:*

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

- **3-CNF** is NP-Complete

Clique

Clique Problem:

– Undirected graph G = (V, E)

– **Clique:** a subset of vertices in V all connected to each other by edges in E (i.e., forming a complete graph)

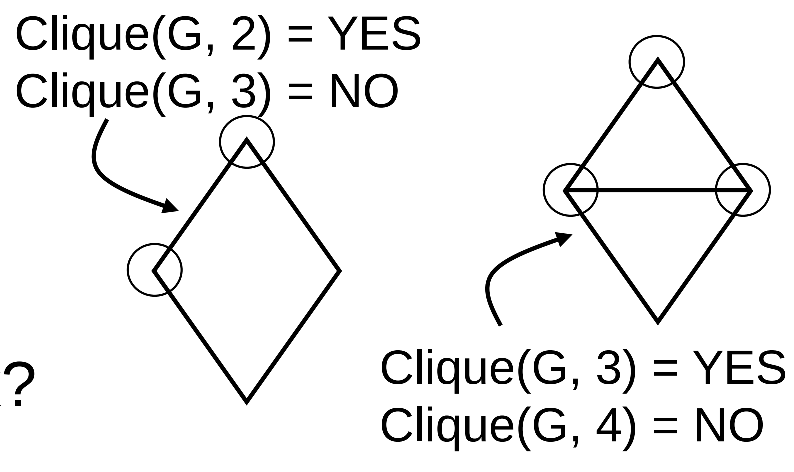
– **Size of a clique:** number of vertices it contains

Optimization problem:

– Find a clique of maximum size

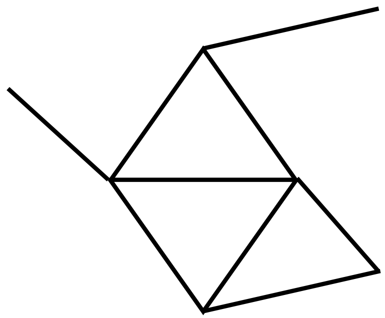
Decision problem:

– Does G have a clique of size k?



Clique Verifier

- **Given**: an undirected graph G = (V, E)
- **Problem**: Does G have a clique of size k?
- **Certificate**:
  – A set of k nodes



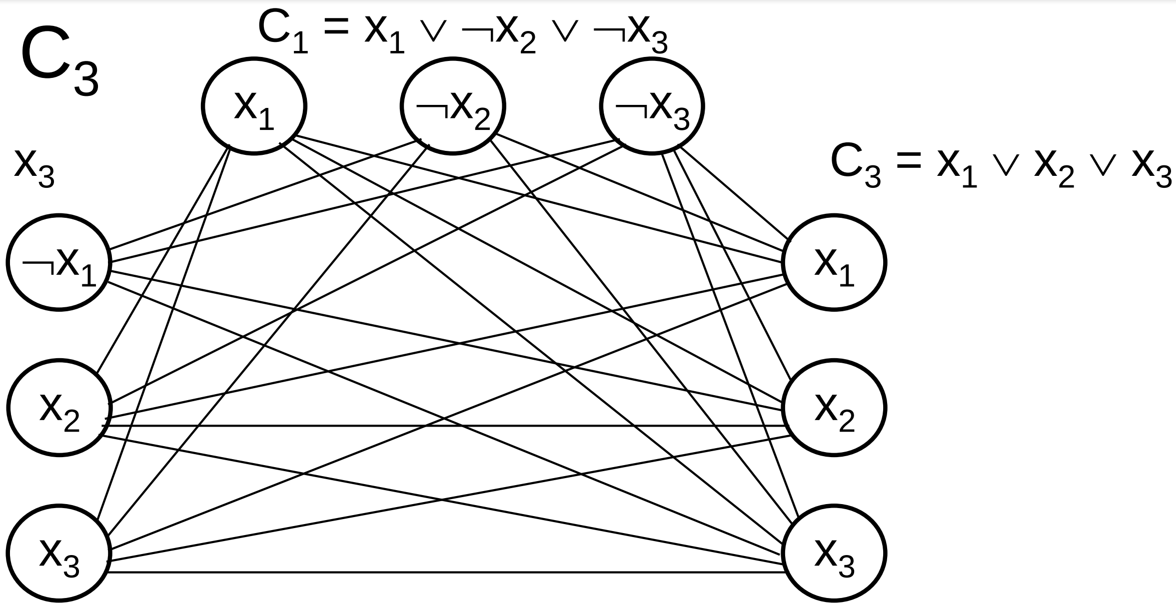
- **Verifier**:
  – Verify that for all pairs of vertices in this set there exists an edge in E

3-CNF $\leq_p$ Clique

- Start with an instance of 3-CNF:
  - $\Phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ (k clauses)
  - Each clause C_r has three literals: $C_r = l_1^r \vee l_2^r \vee l_3^r$
- **Idea:**
  - Construct a graph G such that Φ is satisfiable only if G has a clique of size k

3-CNF $\leq_p$ Clique

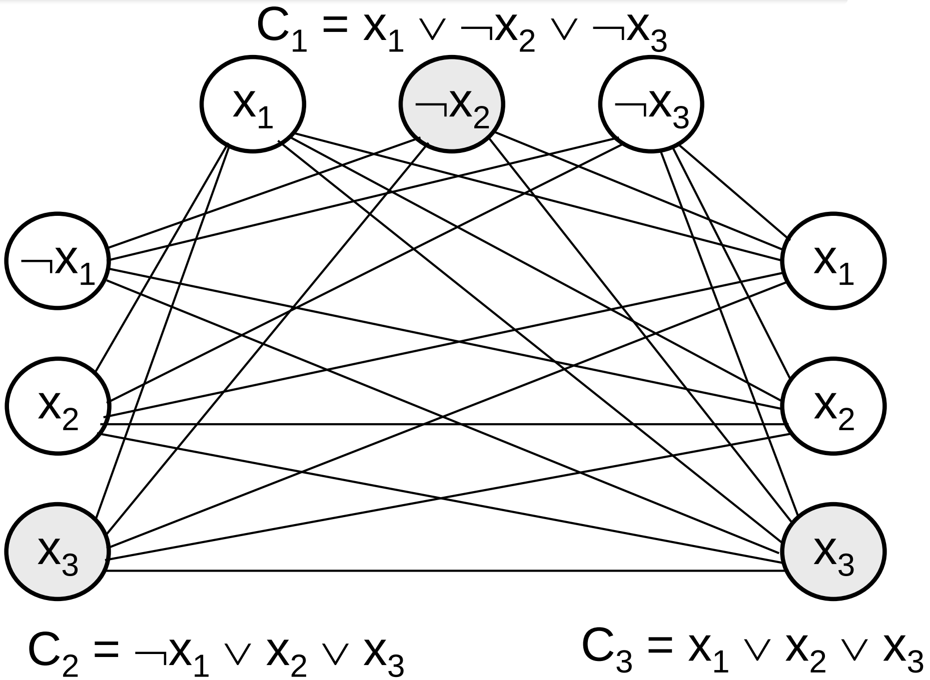
$\Phi = C_1 \wedge C_2 \wedge C_3$



- For each clause $C_r = l_1^r \vee l_2^r \vee l_3^r$ place a triple of vertices v_1^r, v_2^r, v_3^r in V
- Put an edge between two vertices v_i^r and v_j^s if:
  - v_i^r and v_j^s are in different triples
  - l_i^r is not the negation of l_j^s (consistent correspondent literals)

3-CNF $\leq_p$ Clique

$\Phi = C_1 \wedge C_2 \wedge C_3$

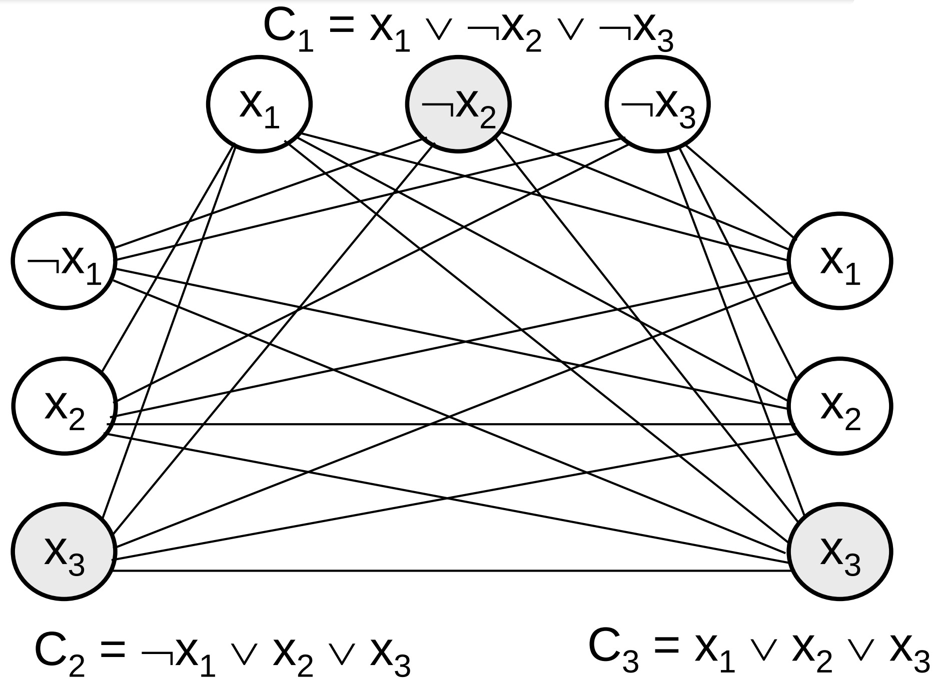


- Suppose Φ has a satisfying assignment
  - Each clause C_r has a literal assigned to 1 – this corresponds to a vertex v_i^r
  - Picking one such literal from each $C_r \Rightarrow$ a set V' of k vertices
- Claim: V' is a clique
  - $\forall$ v_i^r, $v_j^s \in$ V' the corresponding literals are 1 $\Rightarrow$ cannot be complements
  - by the design of G the edge $(v_i^r, v_j^s) \in E$

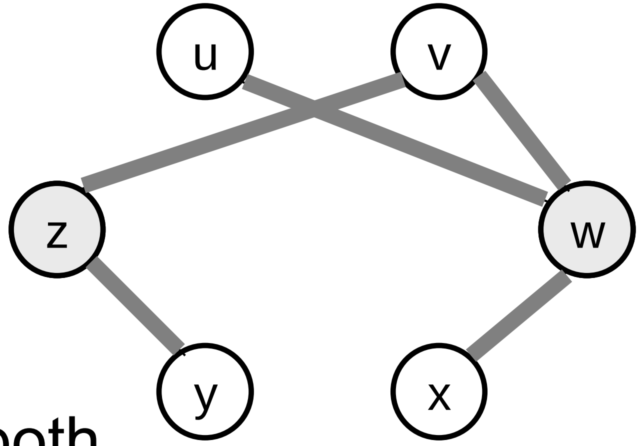
3-CNF $\leq_p$ Clique

$\Phi = C_1 \wedge C_2 \wedge C_3$

- Suppose *G* has a clique of size k
  - No edges between nodes in the same clause
  - Clique contains only one vertex from each clause
  - Assign 1 to vertices in the clique
  - The literals of these vertices cannot belong to complementary literals
  - Each clause is satisfied $\Rightarrow$ Φ is satisfied



Vertex Cover

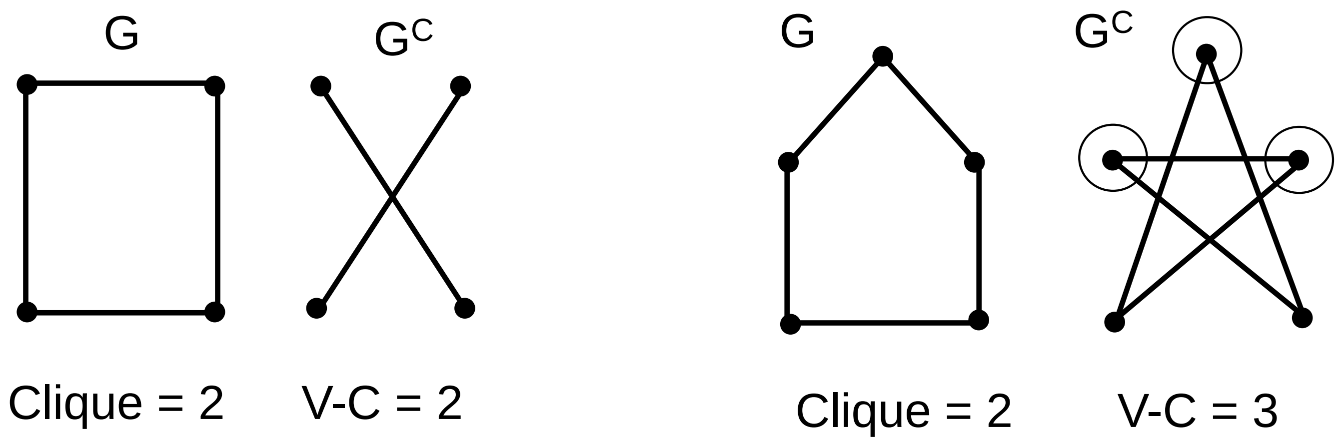


- G = (V, E), undirected graph
- **Vertex cover** = a subset $V' \subseteq V$ such that covers all the edges
  - if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both.
- **Size** of a vertex cover = number of vertices in it

Problem:

- Find a vertex cover of minimum size
- Does graph G have a vertex cover of size k?

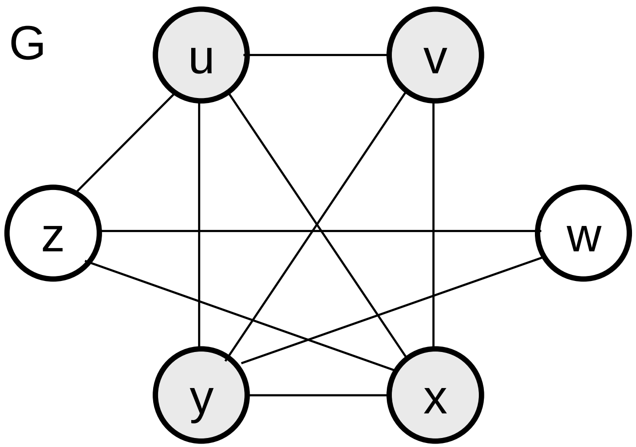
Clique $\leq_p$ Vertex Cover

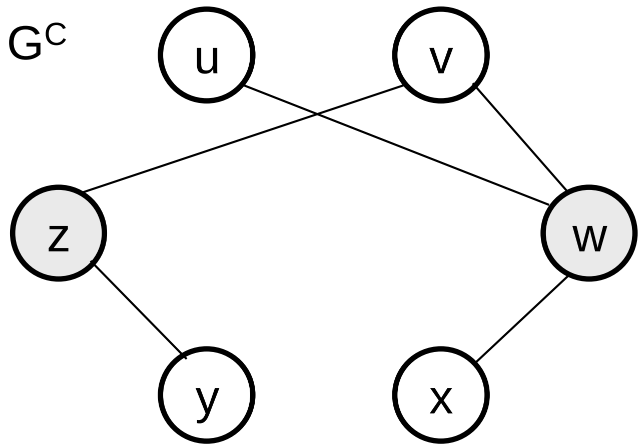


$$\text{Size[Clique]}(G) + \text{Size[V-C]}(G^C) = n$$

- G has a **clique** of size k $\Leftrightarrow$ G^C has a **vertex cover** of size n – k
- S is a clique in G $\Leftrightarrow$ V – S is a vertex cover in G^C

Clique $\leq_p$ Vertex Cover





- $G = (V, E) \Rightarrow G^C = (V, E^C)$

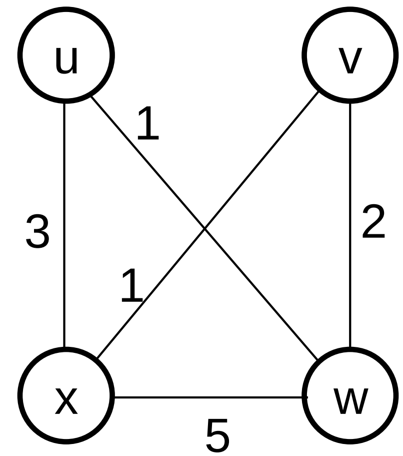
  $E^C = \{(u, v):, u, v \in V, \text{ and } (u, v) \notin E\}$

Idea:

$\langle G, k \rangle$ (clique) $\rightarrow \langle G^C, |V|\text{-}k \rangle$ (vertex cover)

The Traveling Salesman Problem

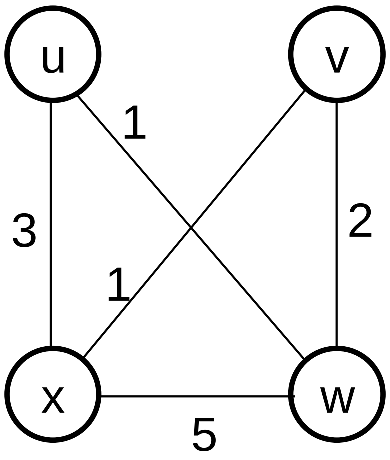
- $G = (V, E)$, $|V| = n$, vertices represent cities
- **Cost**: $c(i, j)$ = cost of travel from city i to city j
- **Problem**: salesman should make a tour (hamiltonian cycle):
  - Visit each city only once
  - Finish at the city he started from
  - Total cost is minimum
- TSP = tour with cost at most k



$\langle u, w, v, x, u \rangle$

TSP $\in$ NP

- **Certificate:**
  - Sequence of n vertices
- **Verification:**
  - Each vertex occurs only once
  - Sum of costs is at most k



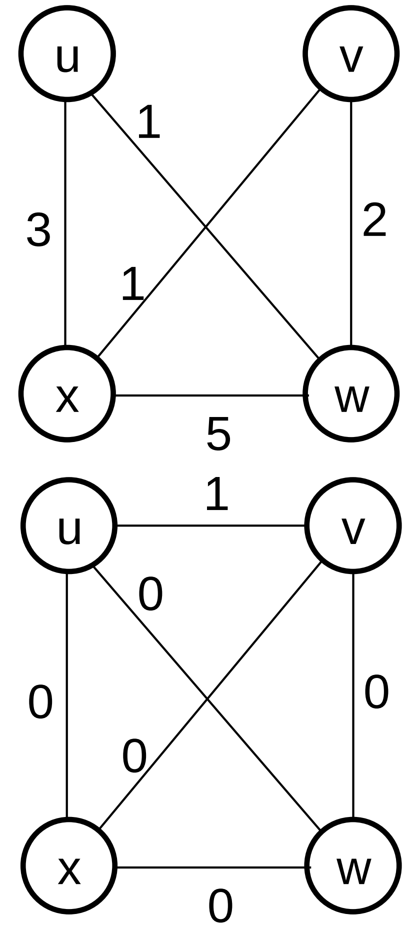
HAM-CYCLE $\leq_p$ TSP

- Start with an instance of Hamiltonian cycle $G = (V, E)$
- Form the complete graph $G' = (V, E')$

$$E' = \{(i, j): i, j \in V \text{ and } i \neq j\}$$

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

- TSP: $\langle G', c, 0 \rangle$
- G has a hamiltonian cycle $\Leftrightarrow$ has a tour of cost at most 0



G'

HAM-CYCLE $\leq_p$ TSP

- G has a hamiltonian cycle h
  - $\Rightarrow$ Each edge in $h \in E \Rightarrow$ has cost 0 in G'
  - $\Rightarrow$ h is a tour in G' with cost 0
- G' has a tour h' of cost at most 0
  - $\Rightarrow$ Each edge on tour must have cost 0
  - $\Rightarrow$ h' contains only edges in E

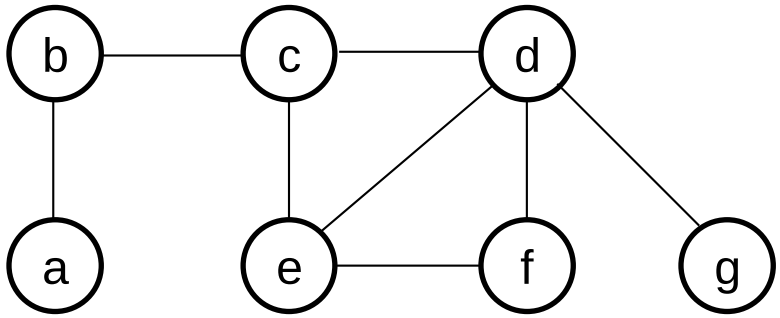
Approximation Algorithms

Ways to get around NP-completeness:

1. If inputs are small, an algorithm with exponential time may be satisfactory
2. Isolate special cases, solvable in polynomial time
3. Find near-optimal solutions in polynomial time
   - Approximation algorithms

The Vertex-Cover Problem

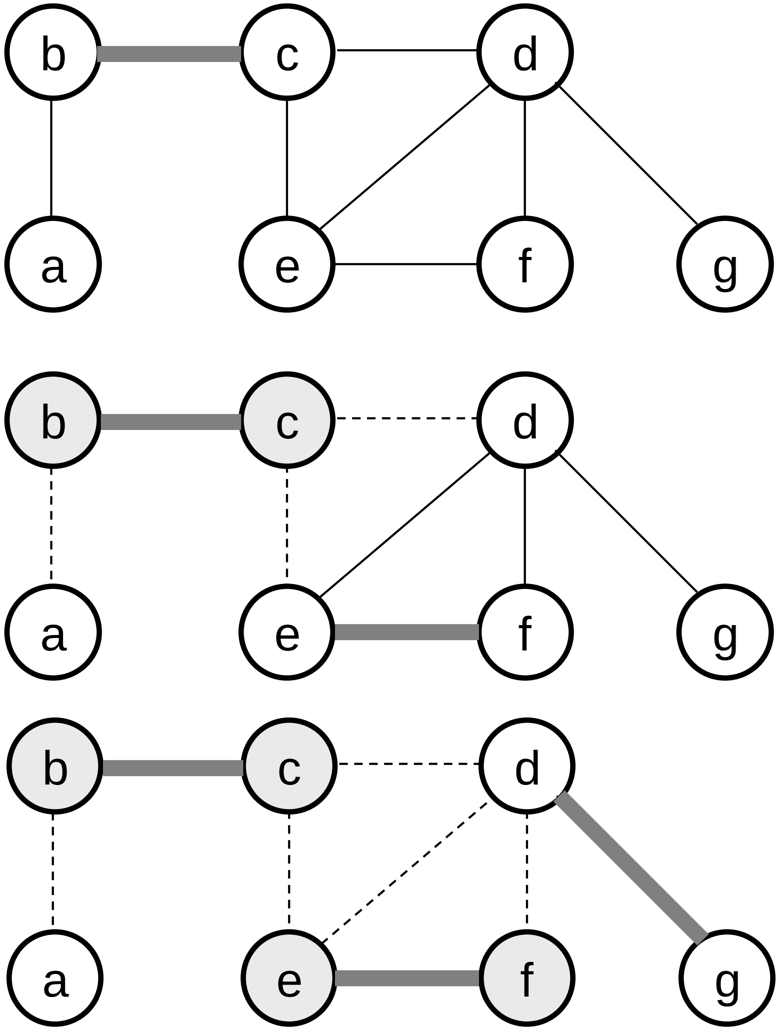
- Vertex cover of G = (V, E), undirected graph
  - A subset V’ $\subseteq$ V that covers all the edges in G



- **Idea**:
  - Repeatedly pick an arbitrary edge (u, v)
  - Add its endpoints u and v to the vertex-cover set
  - Remove all edges incident on u or v

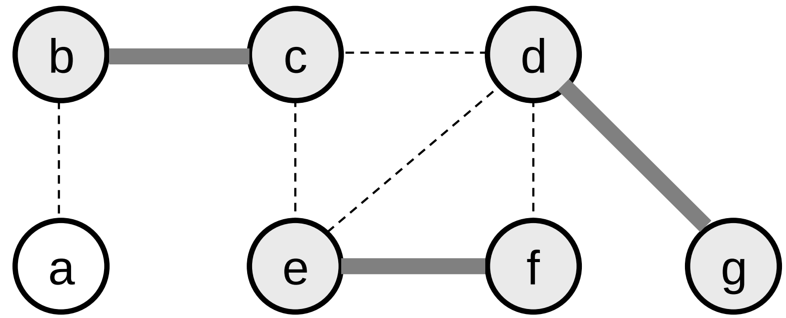
APPROX-VERTEX-COVER(G)

1. $C \leftarrow \emptyset$
2. $E' \leftarrow E[G]$
3. **while** $E' \neq \emptyset$
4. **do** choose (u, v) arbitrary from E'
5. $C \leftarrow C \cup \{u, v\}$
6. remove from E' all edges incident on u, v
7. **return** C

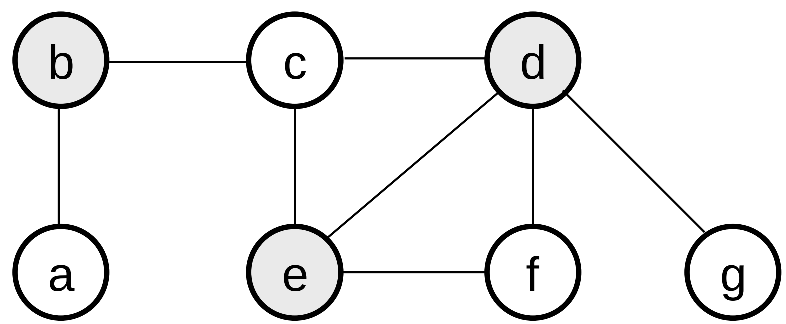


APPROX-VERTEX-COVER(G)

APPROX-VERTEX-COVER:



Optimal VERTEX-COVER:

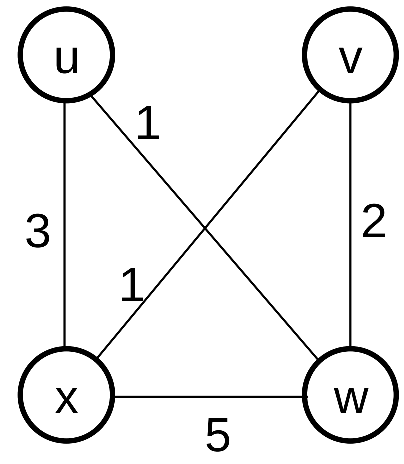


The approximation algorithm returns an optimal solution that is no more than twice the optimal vertex cover

The Traveling Salesman Problem

- G = (V, E), |V| = n, *c(i, j)* = cost of travel from city i to city j
- **Problem**: salesman should make a tour (hamiltonian cycle):
  - Visit each city only once
  - Finish at the city he started from
  - Total cost is minimum
- TSP = tour with cost at most k
- Triangle inequality

$$c(u, w) \le c(u, v) + c(v, w)$$

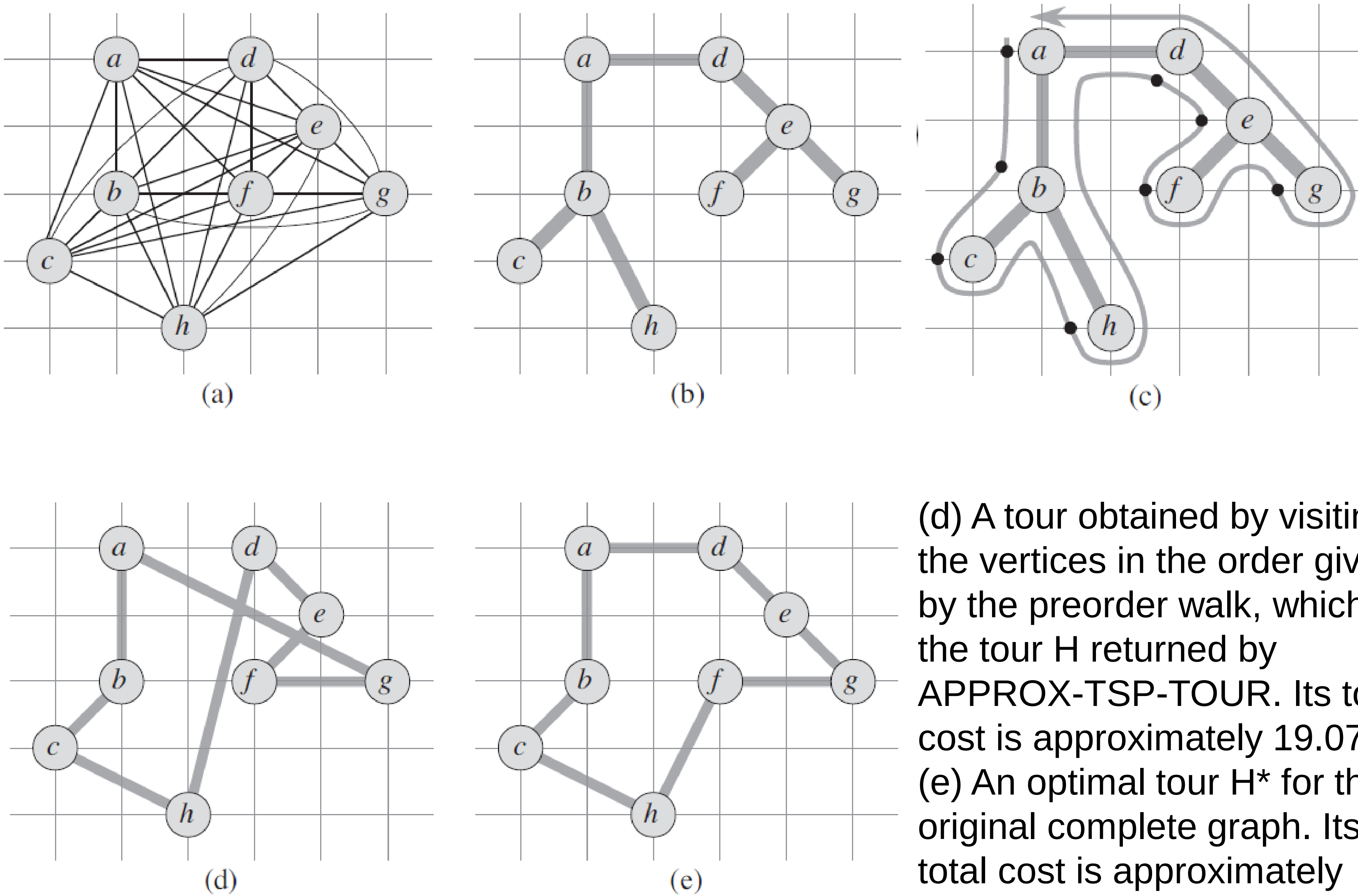


$\langle u, w, v, x, u \rangle$

APPROX-TSP-TOUR

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a “root” vertex
2 compute a minimum spanning tree T for G from root r
  using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited
  in a preorder tree walk of T
4 **return** the hamiltonian cycle H



(d) A tour obtained by visiting the vertices in the order given by the preorder walk, which is the tour H returned by APPROX-TSP-TOUR. Its total cost is approximately 19.074. (e) An optimal tour H* for the original complete graph. Its total cost is approximately 14.715.

The Set Covering Problem

- Finite set X
- Family $\mathcal{F}$ of subsets of X: $\mathcal{F} = \{S_1, S_2, \ldots, S_n\}$

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

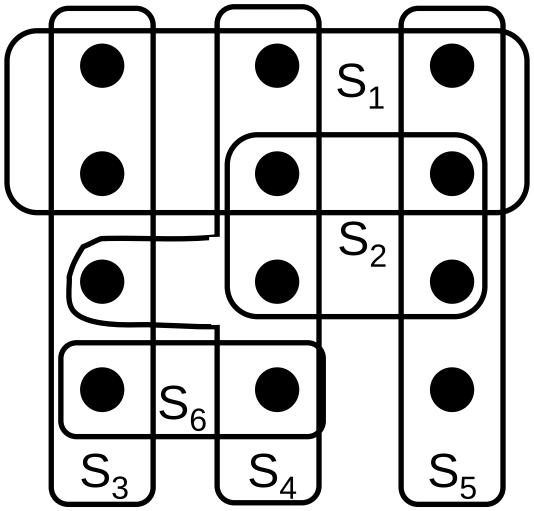
- Find a minimum-size subset $C \subseteq \mathcal{F}$ that covers all the elements in X
- Decision: given a number k find if there exist k sets $S_{i1}, S_{i2}, \ldots, S_{ik}$ such that:

$$S_{i1} \cup S_{i2} \cup \ldots \cup S_{ik} = X$$

Greedy Set Covering

Idea:

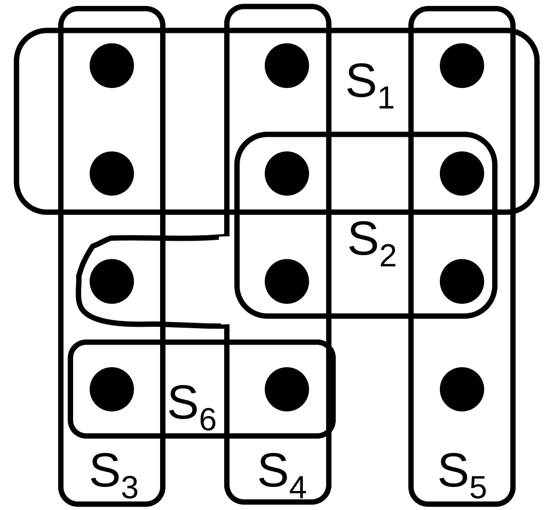
- At each step pick a set S that covers the greatest number of remaining elements

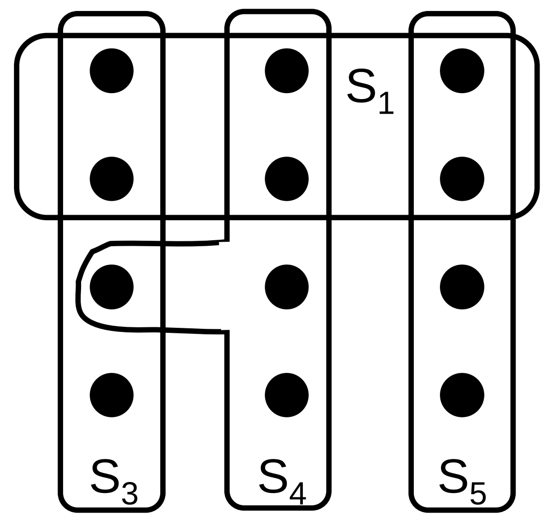


Optimal: C = {S_3, S_4, S_5}

GREEDY-SET-COVER(X, $\mathcal{F}$)

1. $U \leftarrow X$
2. $C \leftarrow \emptyset$
3. **while** $U \neq \emptyset$
4. **do** select an $S \in F$ that maximizes $|S \cap U|$
5. $U \leftarrow U - S$
6. $C \leftarrow C \cup \{S\}$
7. **return** C





Readings

- Chapter 34