CSE 6140/ CX 4140
Computational Science and Engineering
ALGORITHMS

NP Completeness 2

Instructor: Xiuwei Zhang
Assistant Professor
School of Computational Science and Engineering

Based on slides by Prof. Ümit V. Çatalyürek

Summary of last lecture

- optimization problems
- decision problems
  - Solving a problem — Class P: a problem can be solved in polynomial time
  - Verify a candidate solution — Class NP: we can verify a candidate solution in polynomial time

Verifying a Candidate Solution vs. Solving a Problem

- Intuitively it seems much harder (more time consuming) in some cases to **solve** a problem from scratch than to **verify** that a candidate solution actually solves the problem.

  - If there are many candidate solutions to check, then even if each individual one is quick to check, overall it can take a long time

Is P = NP?

- Any problem in P is also in NP:

$$P \subseteq NP$$

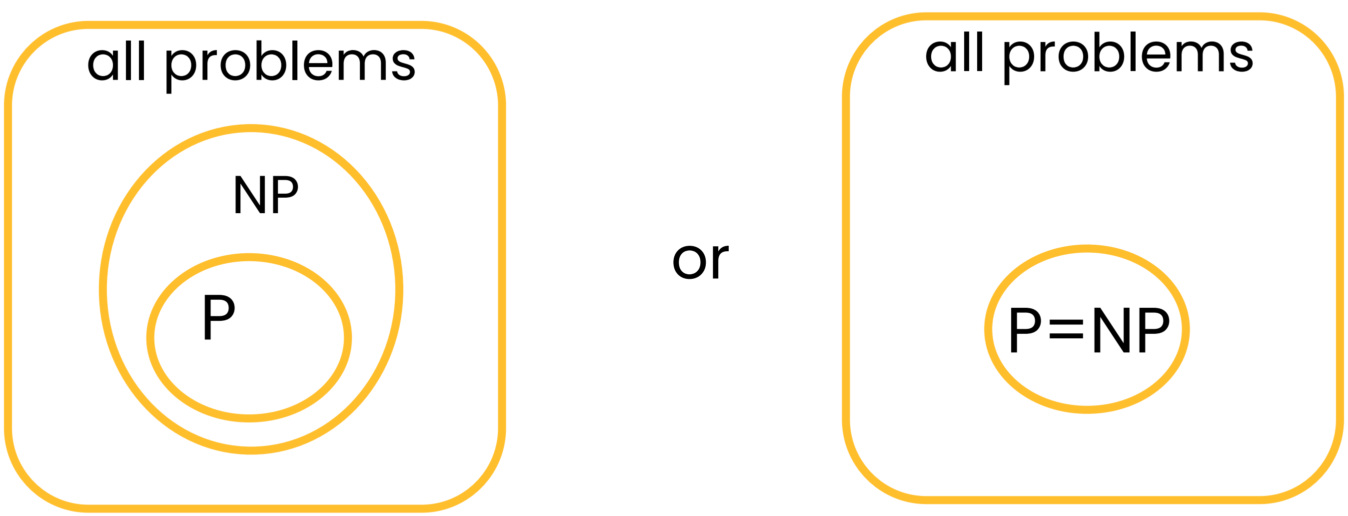
- The big (and **open question**) is whether $NP \subseteq P$ or $P = NP$
  - i.e., if it is always easy to check a solution, should it also be easy to find a solution?

Is P = NP?

- Any problem in P is also in NP:

$$P \subseteq NP$$

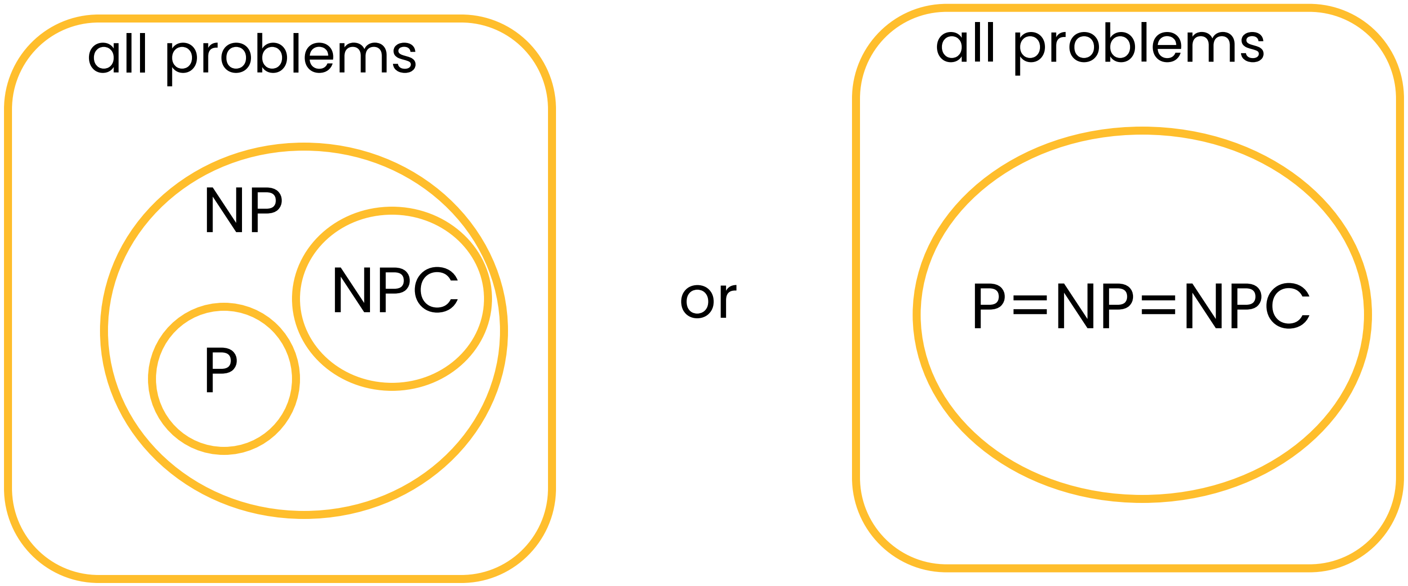
- The big (and **open question**) is whether NP ⊆ P or P = NP
  - i.e., if it is always easy to check a solution, should it also be easy to find a solution?
- Most computer scientists believe that this is false but we do not have a proof …



NP-Complete Problems

- NP-complete problems is class of "hardest" problems in NP.
- If you can solve an NP-complete problem, then you can solve all NP problems (show later).
- Hence, if any NP-complete problem can be solved in poly time, then all problems in NP can be, and thus P = NP.
- Precise definition coming later...

Possible Worlds



NPC = NP-complete

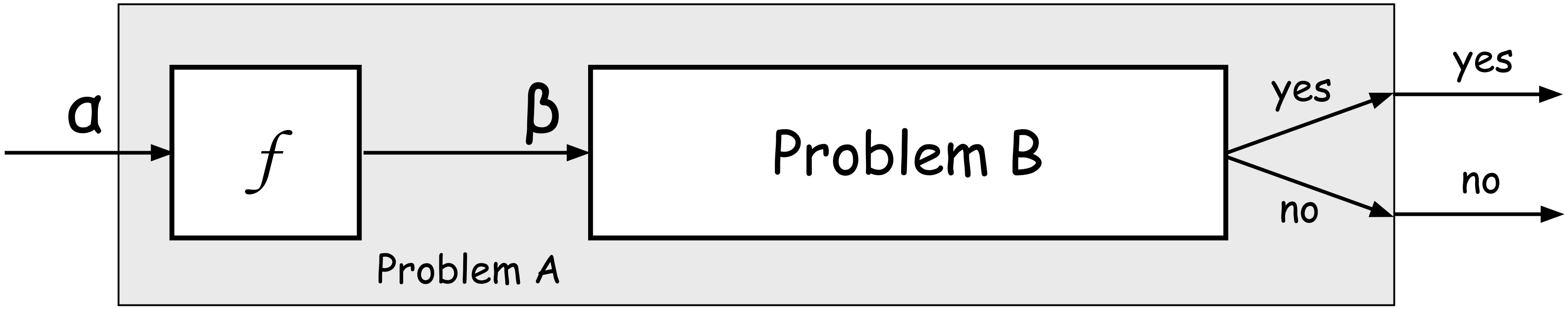
Reductions

- Reduction from A to B is showing that we can solve A using the algorithm that solves B
- We say that <u>problem A is easier than problem B</u>, (i.e., we write **"A ≤ B"**)

Reductions

- **"A ≤ B":** Reduction from A to B is showing that we can solve A using the algorithm that solves B

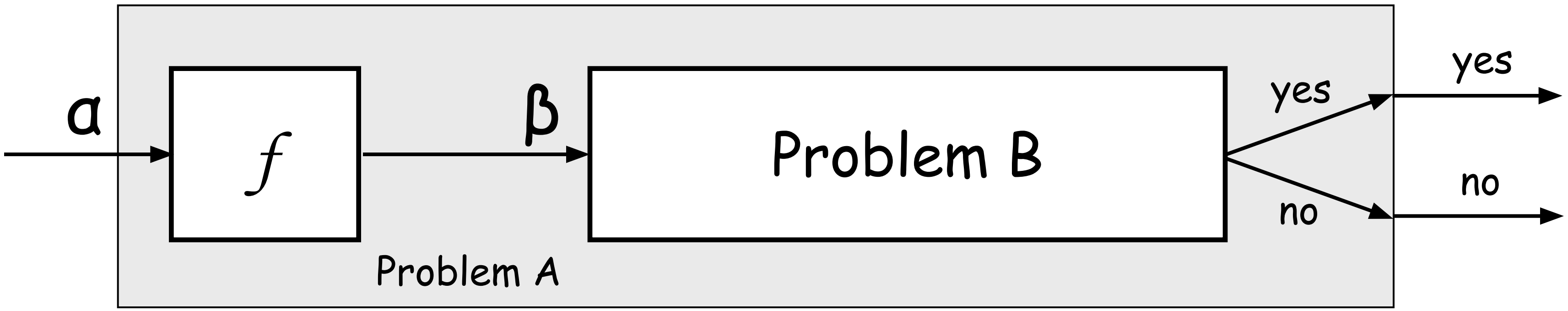
- If we have an oracle for solving B, then we can solve A by making polynomial number of computations and <u>polynomial number of calls to the oracle</u> for B (Cook)
- **Idea:** transform the inputs of A to inputs of B (<u>single call to oracle</u>) (Karp)



Have we already done reductions in class?

- All pairs shortest path: multiple calls to Dijkstra
- K-clustering: use of MST

- We can also do reductions on poly time algorithms

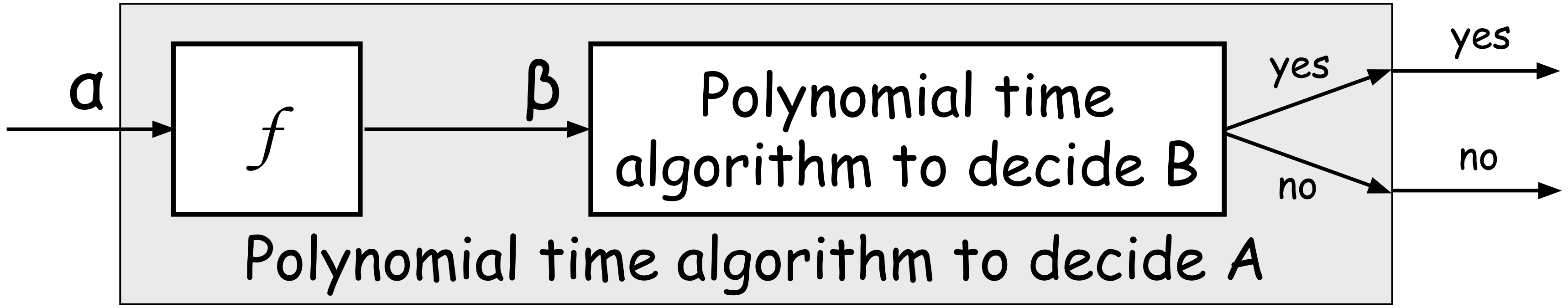


Polynomial Reductions

- Given two problems A, B, we say that A is polynomially **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. $A(i) = YES \Leftrightarrow B(f(i)) = YES$

Proving 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

(e.g. k-Clustering problem was reduced to MST)

Implications of Polynomial-Time Reductions

- Purpose. Classify problems according to relative difficulty.
- 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.
- Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

up to cost of reduction

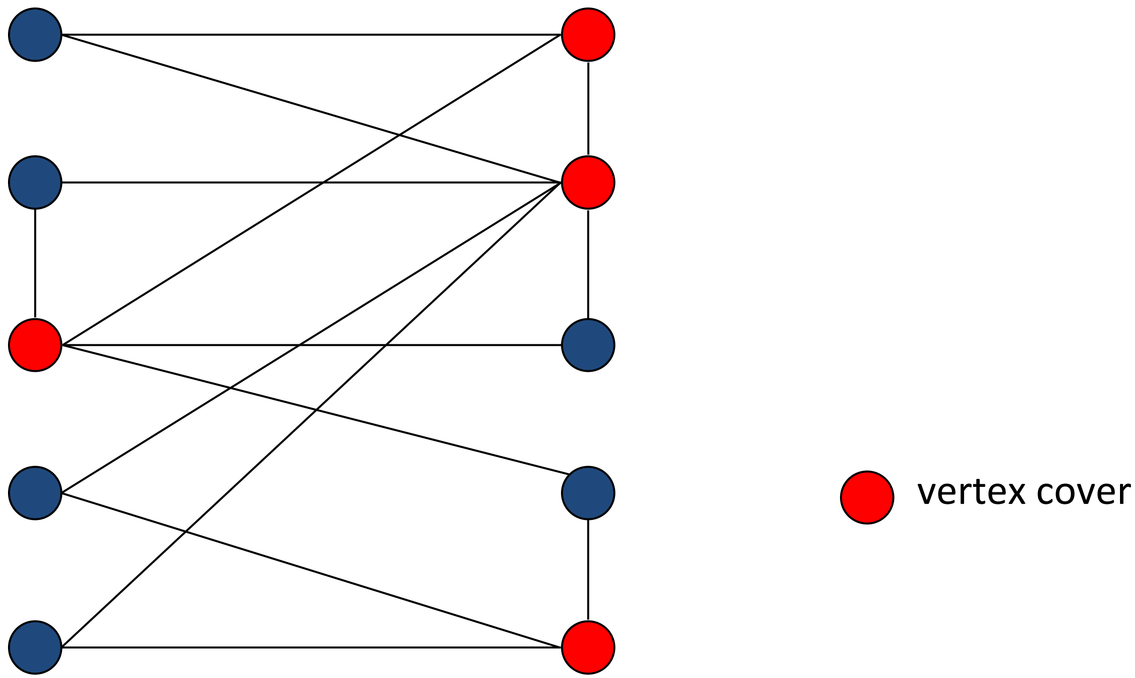
- Transitivity: if $X \leq_p Y$ and $Y \leq_p Z$, then $X \leq_p Z$

Reduction By Simple Equivalence

- Basic reduction strategies.
  - Reduction by simple equivalence.
  - Reduction from special case to general case.
  - Reduction by encoding with gadgets.

Vertex Cover

- MINIMUM VERTEX COVER: Given a graph G = (V, E) , **find the smallest** subset of vertices $S \subseteq V$ such that for each edge at least one of its endpoints is in S?
- VERTEX COVER: Given a graph G = (V, E) and an integer k, **is there a subset** of vertices $S \subseteq V$ such that $\mathbf{|S| \leq k}$, and for each edge, at least one of its endpoints is in S?
- Ex. Is there a vertex cover of size ≤ 4?
- Ex. Is there a vertex cover of size ≤ 3?



Set Cover

- SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq k$ of these sets whose union is equal to U?

- Sample application.
  - m available pieces of software.
  - Set U of n capabilities that we would like our system to have.
  - The *i*th piece of software provides the set $S_i \subseteq U$ of capabilities.
  - Goal: achieve all n capabilities using fewest pieces of software.

Example

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

$k = 2$

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

Vertex cover reduces to set cover

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

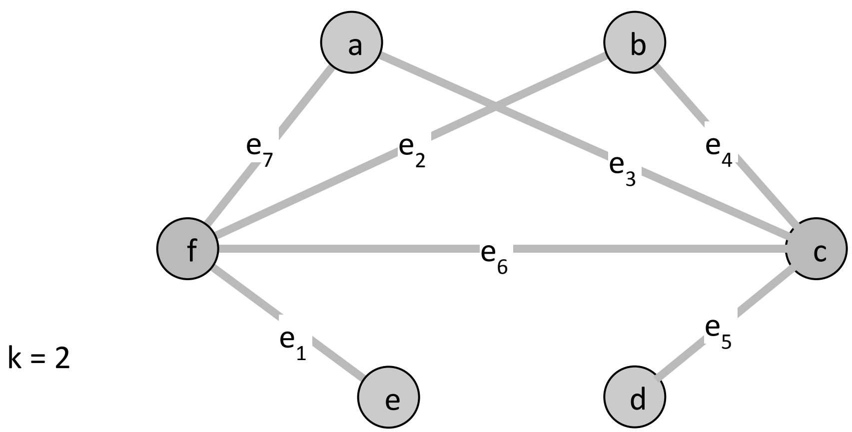
Vertex cover reduces to set cover

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

Pf. Given a VERTEX-COVER instance $G = (V, E)$ and k, we construct a SET-COVER instance (U, S, k) that has a set cover of size k iff G has a vertex cover of size k.

Construction.

- Universe $U = E$.



vertex cover instance
(k = 2)

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

set cover instance
(k = 2)

Vertex cover reduces to set cover

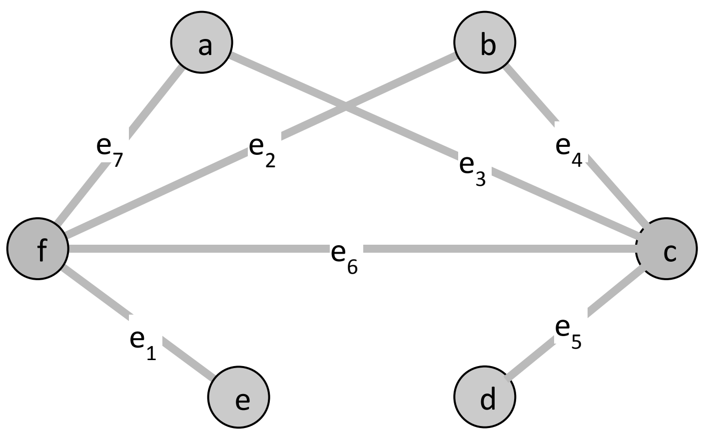
Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

Pf. Given a VERTEX-COVER instance $G = (V, E)$ and k, we construct a SET-COVER instance (U, S, k) that has a set cover of size k iff G has a vertex cover of size k.

Construction.

- Universe $U = E$.
- Create one subset for each node $v \in V$: $S_v = \{e \in E : e \text{ incident to } v\}$.

Show that the reduction algorithm is polynomial



k = 2

vertex cover instance (k = 2)

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

$S_a = \{ 3, 7 \}$ $\quad$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $\quad$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $\quad$ $S_f = \{ 1, 2, 6, 7 \}$

set cover instance (k = 2)

Vertex cover reduces to set cover

Next: show that VC(i)=yes iff SC(f(i))=yes

Vertex cover reduces to set cover

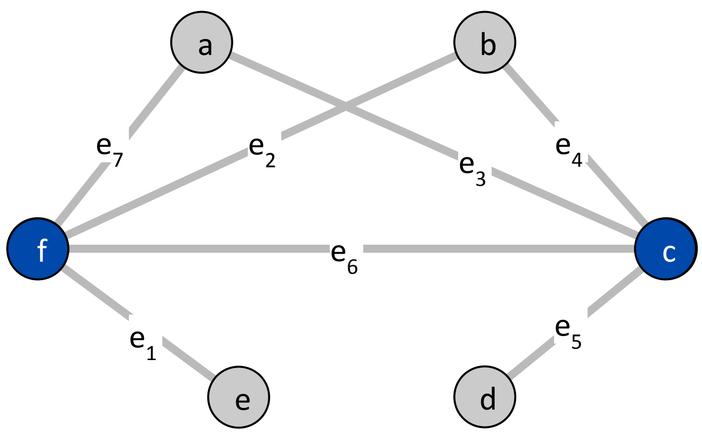
Lemma. $G = (V, E)$ contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

That is, VC(i) = yes $\Leftrightarrow$ SC(f(i)) = yes

Pf. $\Rightarrow$ Let $X \subseteq V$ be a vertex cover of size k in G.

Then $Y = \{ S_v : v \in X \}$ is a set cover of size k. ▪

k = 2



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

$S_a = \{ 3, 7 \}$ $\quad S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $\quad S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $\quad S_f = \{ 1, 2, 6, 7 \}$

vertex cover instance
(k = 2)

set cover instance
(k = 2)

Vertex cover reduces to set cover

VC(i) = yes $\Longrightarrow$ SC(f(i)) = yes

VC(i) is a yes instance $\Longrightarrow$ it has a solution let $V' \subseteq V$ be such a solution

$|V'| \leq k$, every edge has at least one end point in V'

$V' = \{i_1, i_2, \cdots, i_l\},\ l \leq k$

Consider $A = \{S_{i_1}, S_{i_2}, \cdots, S_{i_l}\}$

For the sake of contradiction assume A is not a solution to SC(f(i))

Number of sets in A is $l \leq k$ ✅

then it must be the case that $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_l} \neq U$

$\Longrightarrow \exists e \in U$ that is not in $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_l}$

e also corresponds to an edge in VC(i)

$e = (u, v)$, so S_u and S_v are not in A, i.e., S_u, $S_v \notin A$

$\Longrightarrow u, v \notin V'$ (by construction of A)

$e = (u, v)$ would have been not covered by V'

$\rightarrow\leftarrow$ contradicts V' be solution to VC

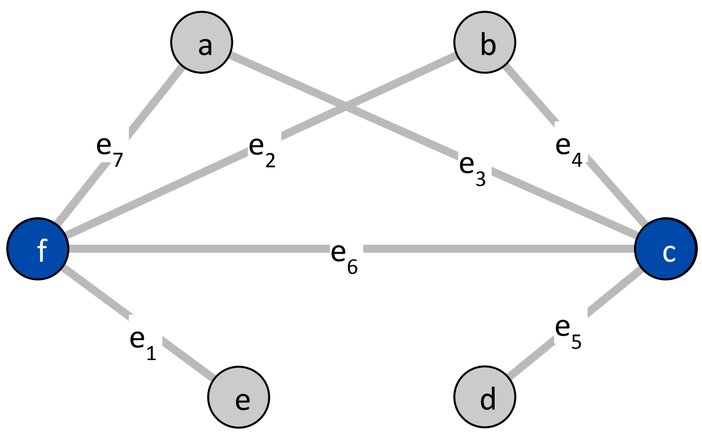
Vertex cover reduces to set cover

Lemma. $G = (V, E)$ contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

Pf. $\Leftarrow$ Let $Y \subseteq S$ be a set cover of size k in (U, S, k).

- Then $X = \{ v : S_v \in Y \}$ is a vertex cover of size k in G. ▪

k = 2



vertex cover instance (k = 2)

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

$S_a = \{ 3, 7 \}$ $\quad S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $\quad S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $\quad S_f = \{ 1, 2, 6, 7 \}$

set cover instance (k = 2)

Vertex cover reduces to set cover

▪ SC(f(i)) = yes $\Longrightarrow$ VC(i) = yes

SC(f(i)) is a yes instance

$\Longrightarrow$ It has a solution and let $A = \{S_{i_1}, S_{i_2}, \cdots, S_{i_l}\}$ be such a solution

$\Longrightarrow$ $l \le k$ and $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_l} = U$ (by definition of SC)

Consider the vertex set $V' = \{i_1, i_2, \cdots, i_l\}$

for the sake of contradiction assume V' is **not** a solution to VC(i)

the number of vertices $l \le k$ ✅

$\Longrightarrow$ it must be breaking the edge covering requirement of VC

$\Longrightarrow$ $\exists e = (u, v) \in E$ such that $u \notin V'$, $v \notin V'$

$\Longrightarrow$ S_u, S_v were not included in solution A

$e = (u, v) \in U$ (by construction of f(i))

S_u, S_v were the only sets that contain e (by construction)

$\Longrightarrow$ $e \notin S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_l}$, i.e., e is not covered by the solution set A,

$\rightarrow\leftarrow$ contradiction with A being solution

Summary

- Problems
  - Decision problems (yes/no)
  - Optimization problems (solution with best score)
- P
  - Decision problems (decision problems) that can be solved in polynomial time
  - Can be solved “efficiently”
- NP
  - Decision problems whose “YES” answer can be verified in polynomial time, if we already have the proof (or witness)

NP-Completeness (formally)

- A problem Y is **NP-hard** if $X \leq_p Y$ for <u>all</u> $X \in$ **NP**
  - A problem is NP-hard iff an polynomial-time algorithm for it implies a polynomial-time algorithm for every problem in NP
  - NP-hard problems are at least as hard as any NP problem
- A problem Y is **NP-complete** if:

  (1) $Y \in$ **NP**

  (2) Y is **NP-hard**

NP-hard problems *do not have to be in NP*, and *they do not have to be decision problems*.