Georgia Tech

CSE 6140/ CX 4140:

Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina, Jennifer Welch, George Bebis, and Kevin Wayne

The class P

- **Class P** consists of (decision) problems that are solvable in polynomial time
- Beware of data size:
  - n encoded in unary if you need to enumerate n objects
  - W encoded in binary if weight or other integer part of the input of the problem
  - *"The composition of two polynomials is a polynomial"*
  - Graph: data in O(n) equivalent to O(n+m), equivalent to O(n^1000) *in terms of data size*

A first example: 2-partition

- 2-PARTITION: Given n positive integers $a_1, \ldots, a_n$, is there a subset I of {1, …, n} such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?
  - $S = \sum_{1 \le i \le n} a_i$
  - Data of size $n \log(S)$, the a_i's are encoded in binary
  - Show that n is in the data size to avoid mistakes
  - Pseudo-polynomial: poly if data encoded in unary

A second example: bipartite graphs

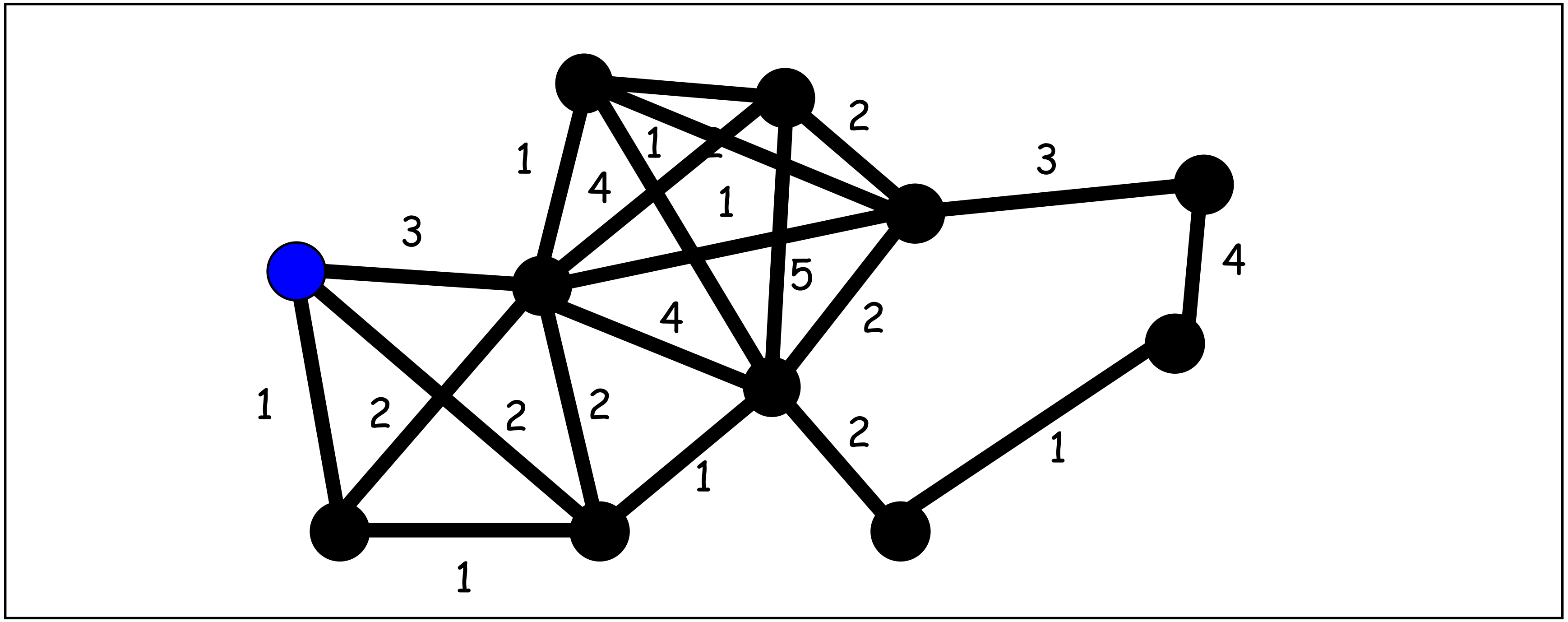
- BIPARTITE: Given a graph G, is G a bipartite graph?
- How do we encode a graph? What is the size of data?
  - Remember: n vertices and m edges
- Which of these are correct?
  1. n + m
  2. n + log(m)
  3. log(n) + log(m)
  4. n
- 1,2,4 are all correct, m is polynomial in n (at most n^2 edges) (for *data size*)
- Need to enumerate all vertices to describe the pb instance, so 3 is not correct
- Greedy algorithm polynomial in n (good practice problem!), BIPARTITE is in P

Tractable/Intractable Problems

- Problems in P are also called **tractable**
- Problems **not** in P are **intractable**

- Are non-polynomial algorithms always worst than polynomial algorithms?
  - $n^{1,000,000}$ is *technically* tractable, but really impossible
  - $n^{\log \log \log n}$ is *technically* intractable, but easy

Example: TSP



- For each two cities, an integer cost is given to travel from one of the two cities to the other. The salesperson wants to make a minimum cost circuit visiting each city exactly once.
- TSP: Given a complete graph G=(V,E), a cost function w:E->N, and an integer k, is there a cycle C going through each vertex once and only once, with $\sum_{e \in C} w(e) \leq k$?

The class NP

- NP is the class of problems for which a candidate solution can be verified in polynomial time

- NP does not stand for not-P!!
- NP='nondeterministic polynomial'

- P is a subset of NP

Nondeterministic and NP Algorithms

Nondeterministic algorithm = two stage procedure:

1) Nondeterministic ("guessing") stage:

   generate randomly an arbitrary 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

NP algorithms (Nondeterministic polynomial)

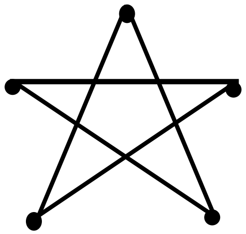
verification stage is polynomial

Verifying a Candidate Solution

- Difference between solving a problem and verifying a candidate solution:
- Solving a problem: is there a path in graph G from vertex u to vertex v with at most k edges?
- Verifying a candidate solution: is $v_0, v_1, \ldots, v_\ell$ a path in graph G from vertex u to vertex v with at most k edges?

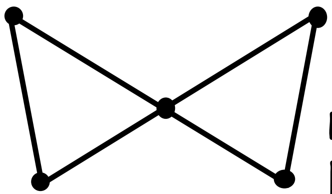
Verifying a Candidate Solution

- A Hamiltonian cycle in an undirected graph is a cycle that visits every vertex exactly once.
- Solving a problem: is there a Hamiltonian cycle in graph G?
- Verifying a candidate solution: is $v_0, v_1, \ldots, v_\ell$ a Hamiltonian cycle of graph G?

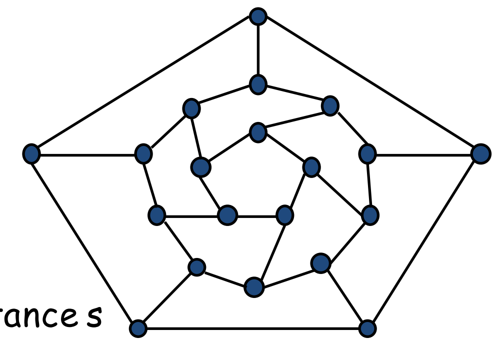


hamiltonian

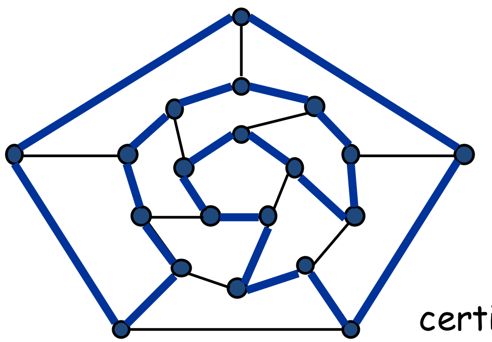
- **Certificate**: A list of n nodes.
- **Certifier**: Check that the list contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.
- **Conclusion**: HAM-CYCLE is in NP.



not hamiltonian



instance s



certificate t

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.

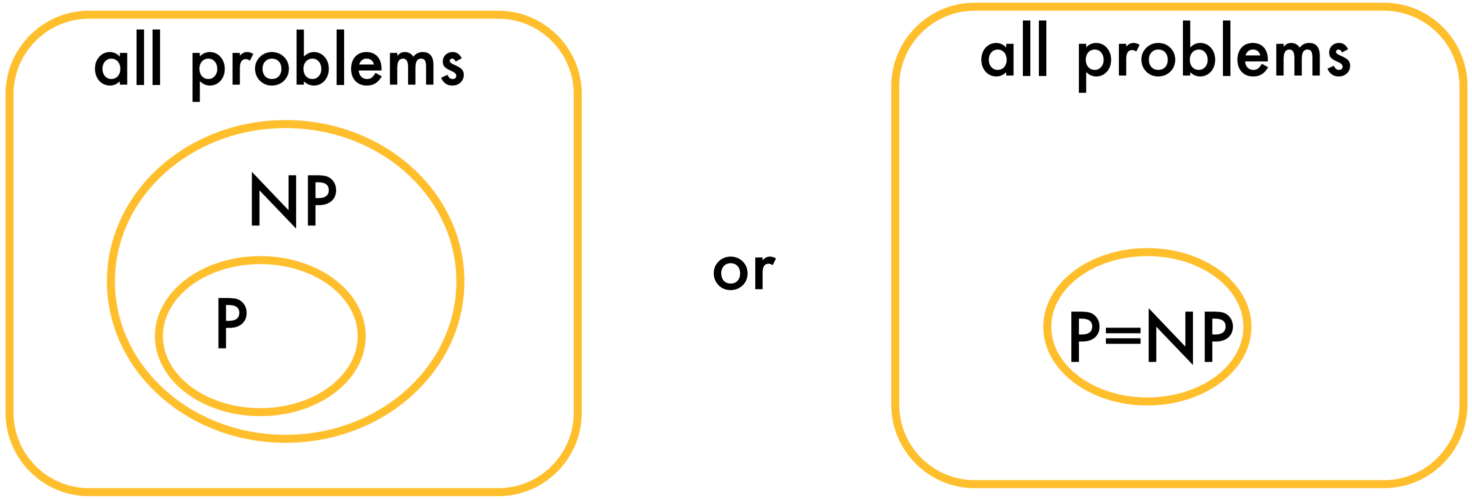
  - If there are many candidate solutions to check, then even if each individual one is quick to check, overall it can take a long time

Is P = NP?

- Any problem in P is also in NP:

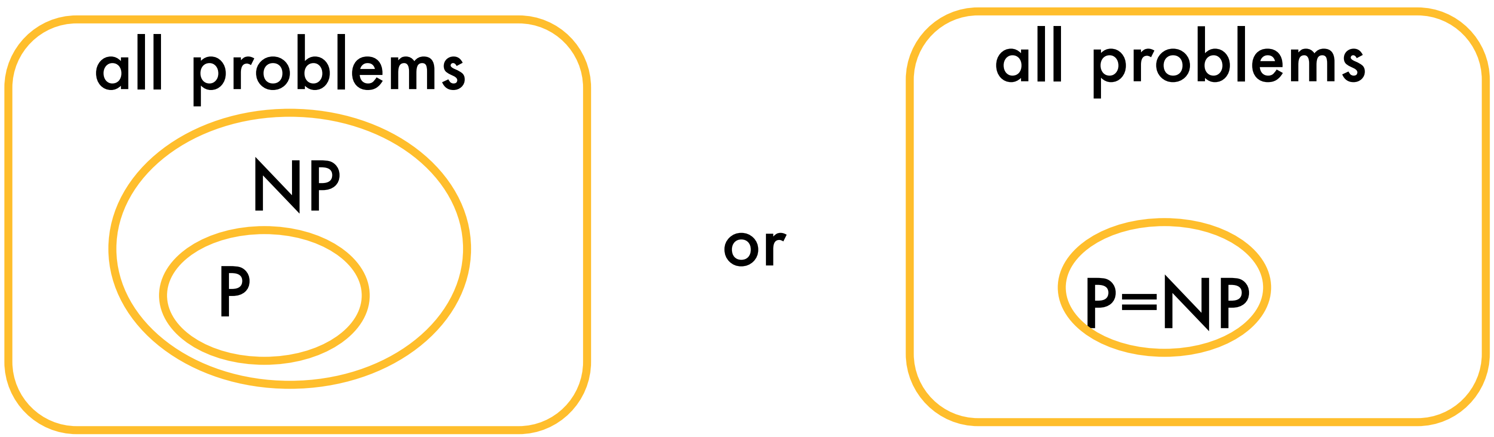
$$P \subseteq NP$$

- 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?
- Most computer scientists believe that this is false but we do not have a proof …



Problems not in NP?

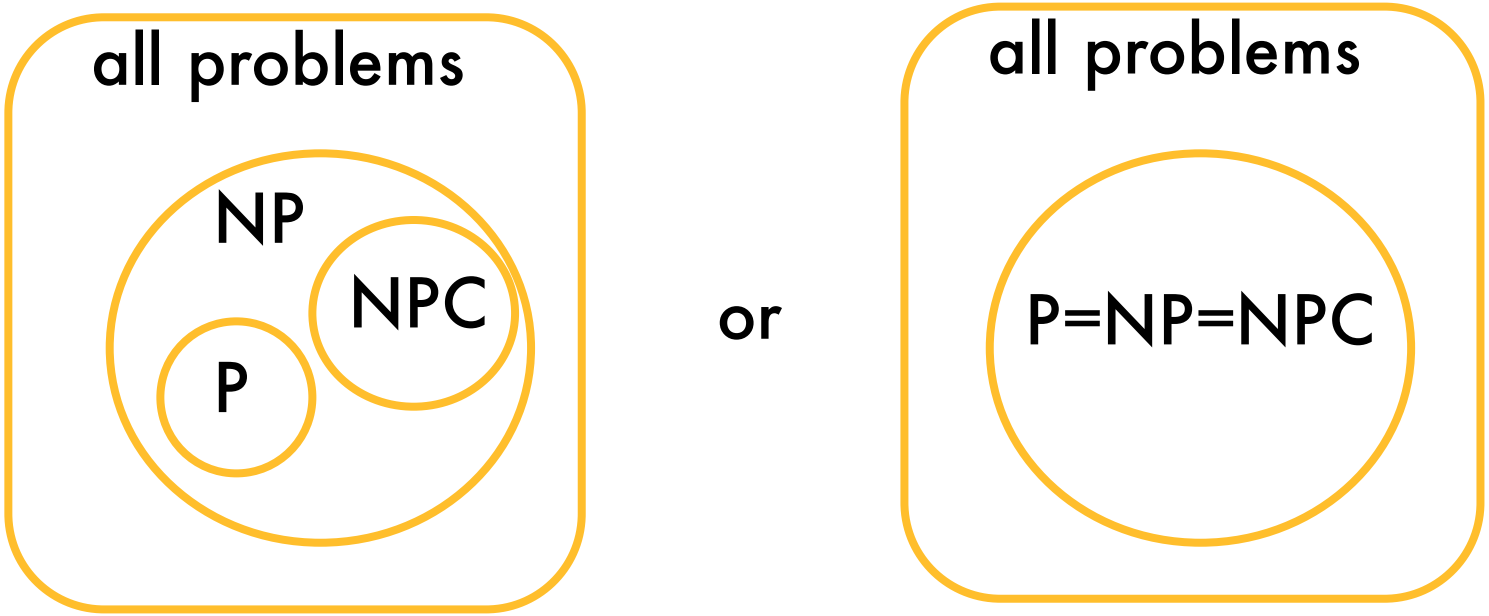
- We have seen several problems in NP: all problems in P, 2-PARTITION, TSP…
- Problems not in NP are rarely encountered but they exist!
  - **Negation of TSP**: Given a problem instance of TSP, is it true that there is no cycle in the graph of length n/2 ?
  - (BTW, what is the input data size for TSP? TSP: Given a complete graph G=(V,E), a cost function w:E->N, and an integer k, is there a cycle C going through each vertex once and only once, with $\sum_{e \in C} w(e) \leq k$?)
  - Difficult to think of a certificate… Open problem whether in NP or not



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...

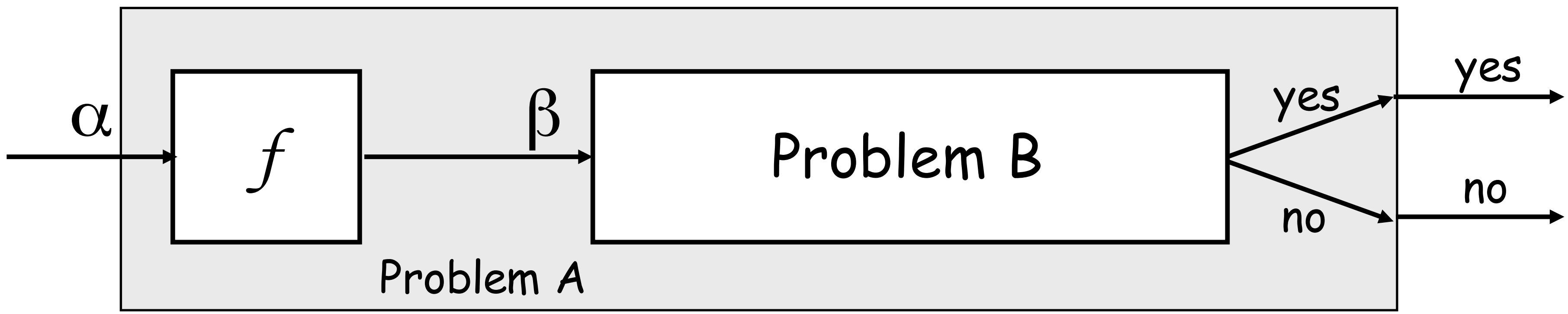
Possible Worlds



NPC = NP-complete

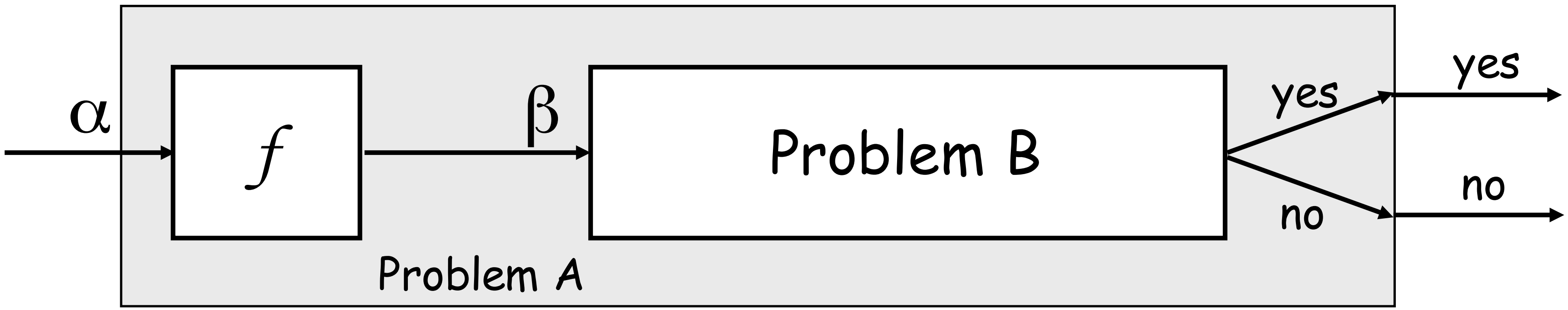
Reductions

- Reduction from A to B is showing that we can solve A using the algorithm that solves B
- We say that problem A is easier than problem B, (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 polynomial number of calls to the oracle for B (Cook)
- **Idea:** transform the inputs of A to inputs of B (single call to oracle) (Karp)



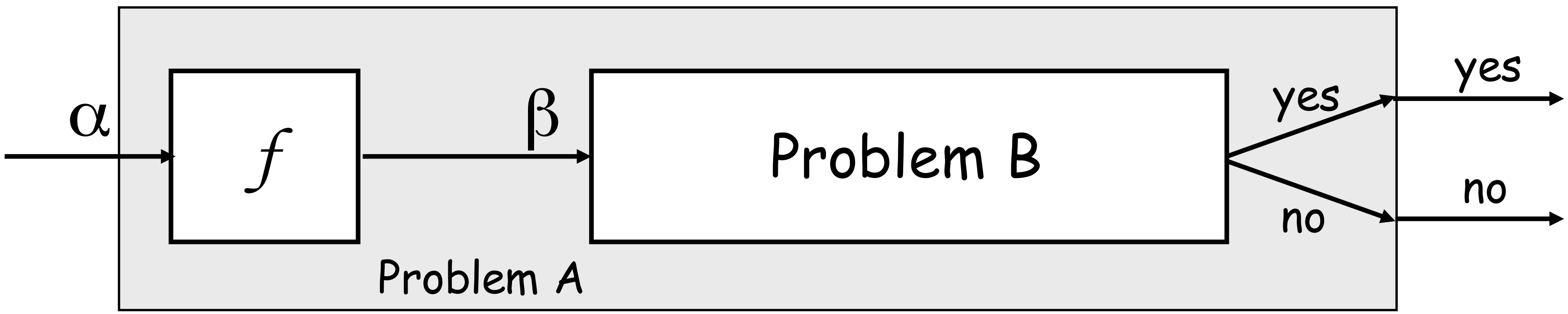
Have we already done reductions in class?

- All-pairs-shortest-paths:

  multiple calls to single-source-shortest-paths

- K-clustering: use of MST

- We can do reductions on poly time algorithms

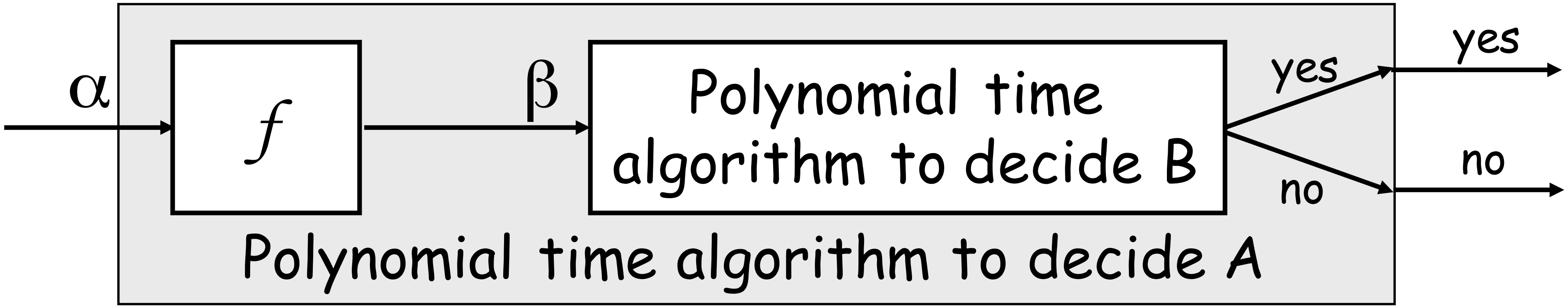


Polynomial Reductions

- 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.

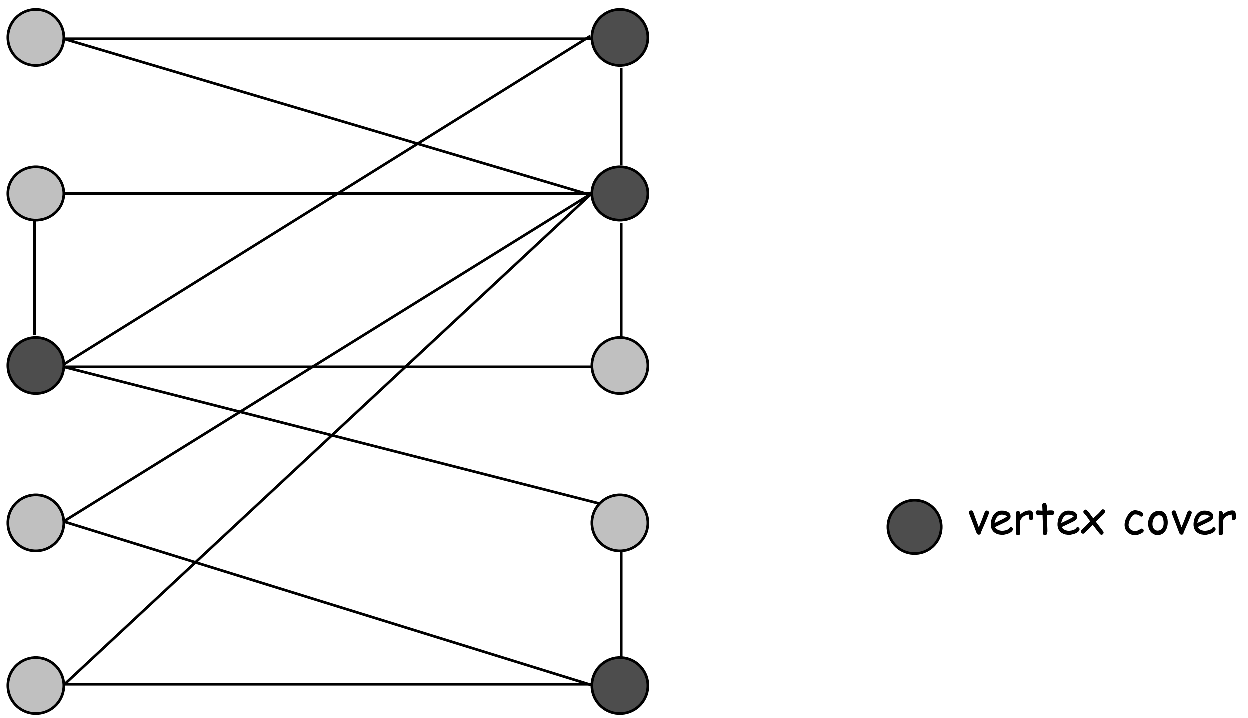
Vertex Cover

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 $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.

Ex. Is there a vertex cover of size ≤ 3? No.



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 ith piece of software provides the set $S_i \subseteq U$ 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 (KT 8.1)

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

Pf. Given a VERTEX-COVER instance {G = (V, E), k}, we construct a SET-COVER instance {U, {S}, k'} whose size equals the size of the vertex cover instance.

Construction. (Proof of correctness to be done next class on paper)

- Create SET-COVER instance:
  - $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$, $k' = k$
- Prove that Set-cover of size $\leq k$ iff vertex cover of size $\leq k$.