NP-completeness Again

CS 3AC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material partially based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 8)

Polynomial-time reductions

- Desiderata. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?
- Reduction. Problem X polynomial-time 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.
- Notation. $X \leq_P Y$.
- Note. We pay for time to write down instances sent to oracle $\Longrightarrow$ instances of Y must be of polynomial size.
- Caveat. Don't mistake $X \leq_P Y$ with $Y \leq_P X$!

Polynomial-time reductions: Design algorithms.

- If $X \leq_P Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time.
- We have already used this type of reduction in this course:
  - Problem 4 (*CluNet*) from Assignment 1 can be reduced to *Stable marriage*, i.e. *CluNet* $\leq_P$ *Stable marriage*
  - *Bipartite Matching* has been reduced to *Max Flow*, i.e. *Bipartite Matching* $\leq_P$ *Max Flow*
  - *"Toy" Airline Scheduling* has been reduced to *Max Flow*, i.e. *"Toy" Airline Scheduling* $\leq_P$ *Max Flow*
  - many other (assignments and other courses)

  In all cases above transformation (cost of reduction) was linear.
- **THIS IS NOT WHAT NEEDS TO BE DONE TO SHOW NP-COMPLETENESS.**

Polynomial-time reductions: Establish intractability.

- 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 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.
- **THIS IS A PROPER WAY TO SHOW NP-COMPLETENESS.**

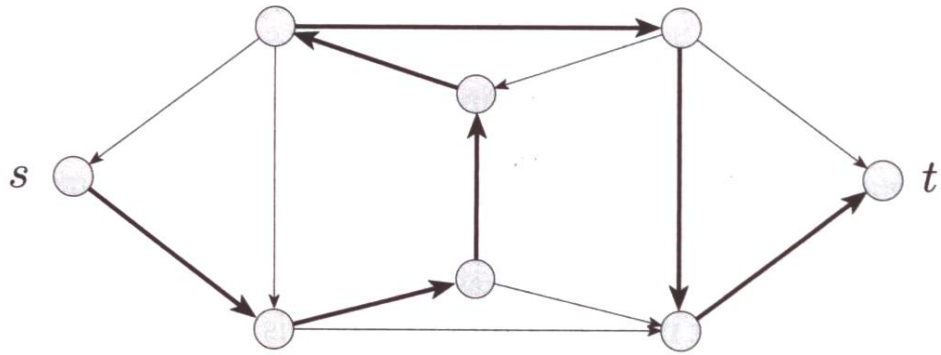
The class **P**

Definition

The class **P** (from *Polynomial*) consists of those problems that are solvable in polynomial time. More specifically, they are problems that can be solved in $O(n^k)$ for some constant k. where n is the size of the input to the problem.

Verifiers

- A **Hamiltonian path** in a directed graph G is a path that goes through each node once.



- **Problem:** Does a directed graph contains a Hamiltonian path connecting two specified nodes?
- Exponential algorithm is easy, check all cases.
- Polynomial algorithm is not found.
- However, for a given path we can *verify* (in $O(n)$ time) if it is Hamiltonian!

Verifiers

- "Does G **have** a Hamiltonian path from s to t?" - is polynomially **verifiable**.
- "Does G **have not** a Hamiltonian path from s to t?" - is **not** polynomially **verifiable**.

Definition

A **verifier** is an algorithm that can verify if a given instance is a solution or not.

The class **NP**

Definition (with verifiers)

The class **NP** (from *Nondeterministic Polynomial*) consists of those problems that are **verifiable** in polynomial time. More specifically, they are problems that can be *verified* in $O(n^k)$ for some constant k. where n is the size of the input to the problem.

Hamiltonian Path is such a problem!

Definition (with *nondeterministic* algorithms)

The class **NP** (from *Nondeterministic Polynomial*) consists of those problems that are *solvable* in polynomial time by *nondeterministic* algorithms. More specifically, they are problems that can be solved in $O(n^k)$ for some constant k. where n is the size of the input to the problem, by *nondeterministic* algorithms.

- The idea of *Nondeterministic Algorithms* is a simple consequence of *angelic* semantics.

NP-completeness

Definition

A problem Y is **NP**-complete if it satisfies the following two conditions:

1. $Y \in \mathbf{NP}$
2. every $X \in \mathbf{NP}$ is *polynomially reducible* to Y, i.e. $X \leq_P Y$.

Let **NPC** denote the class of all **NP**-complete problems.

Theorem

Suppose Y is **NP***-complete. Then* $Y \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.

NP-completeness: basic tool

Theorem

If X is **NP***-complete, $Y \in$* **NP***, and $X \leq_P Y$, then Y is* **NP***-complete.*

Algorithm (Showing NP-completeness of Y)

1. *First show that $Y \in$* **NP***. This is usually done by showing that an instance of Y has a* **polynomial verifier***.*
2. *Find a problem X that has been proven to be* **NP***-complete. For example, 3-SAT, VECTOR-COVER, HAMILTON-CYCLE, etc. If Y is a graph problem, try first X that is also a graph problem.*
3. *Show $X \leq_p Y$, i.e. X can be* **polynomially** *reduced to Y. While the fact that a transformation o X into an* instance *of Y is polynomial is often almost obvious,* **always** *mention it and explain.*

NP-completeness: frequent errors

Algorithm (Showing NP-completeness of Y)

1. *First show that $Y \in$ **NP**.*
2. *Find a problem X that has been proven to be **NP**-complete.*
3. *Show $X \leq_p Y$, i.e. X can be **polynomially** reduced to Y.*

Frequent Errors.

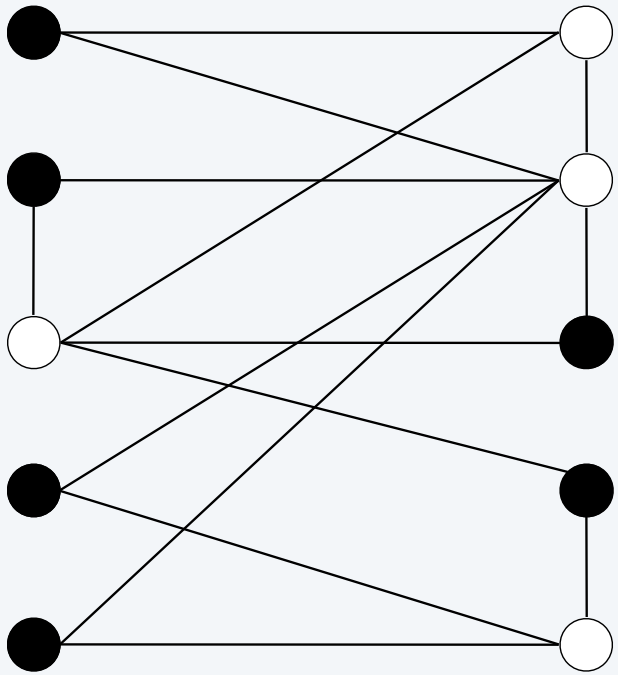
- It is NOT shown that $Y \in$ **NP**. Usually can be fixed.
- It is attempted to show that $Y \leq_P X$ instead of $X \leq_P Y$. This is the most serious error.
- It is not argued that transformation of X into an instance of Y is polynomial. Usually can be fixed.

A proof that INDEPENDENT-SET is **NP**-complete

INDEPENDENT-SET. Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

Ex. Is there an independent set of size ≥ 6?

Ex. Is there an independent set of size ≥ 7?



independent set of size 6

A proof that INDEPENDENT-SET is **NP**-complete

1. We need to show first that INDEPENDENT-SET $\in$ **NP**.

Proposition

INDEPENDENT-SET $\in$ **NP**.

Proof.

It suffices to show that an instance of INDEPENDENT-SET has a polynomial verifier. Let $G = (V, E)$ be a graph. Consider a given set of vertices $S \subseteq V$ with $|S| \geq k$. The brute force algorithm takes each edge $e = (v, w)$ form E and checks if $v \in S$ and $w \in S$. Hence the time complexity is $O(|E||S|) = O(n^2 \cdot n) = O(n^3)$ i.e. it is polynomial. Hence INDEPENDENT-SET $\in$ **NP**. $\square$

2. Assume that we know that 3-SAT is **NP**-complete.
3. We now need to show that

$$\text{3-SAT} \leq_P \text{INDEPENDENT-SET}$$

NOT vice versa!

3-SAT

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. A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given 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).

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

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

3-SAT reduces to INDEPENDENT-SET

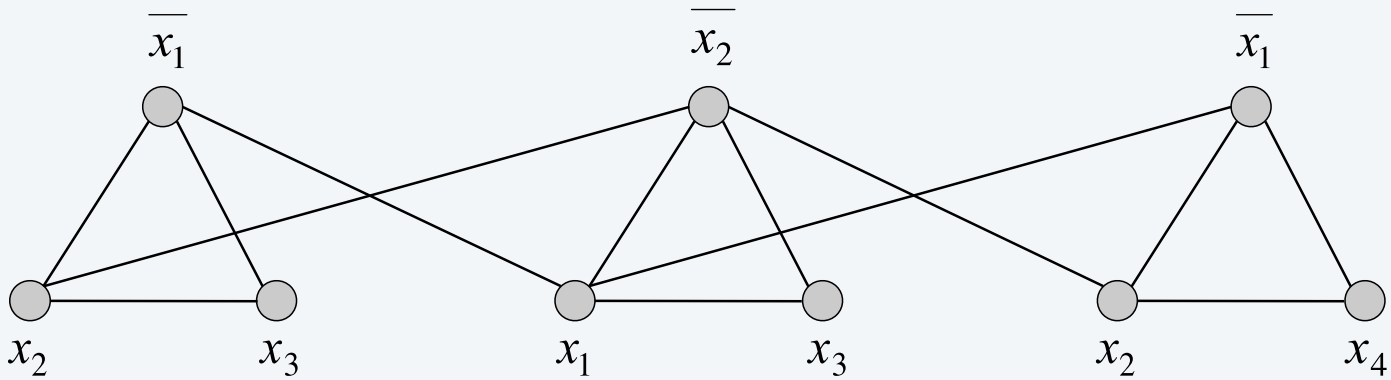
Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

Pf. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 nodes 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

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

Complexity of Reduction

- Constructing k triangles is $O(k)$.
- Connecting literals to their negations is also $O(k)$.
- Hence the reduction is polynomial.
- You do not always be so precise, it often goes without saying, but at least needs to be mentioned.

3-SAT reduces to INDEPENDENT-SET

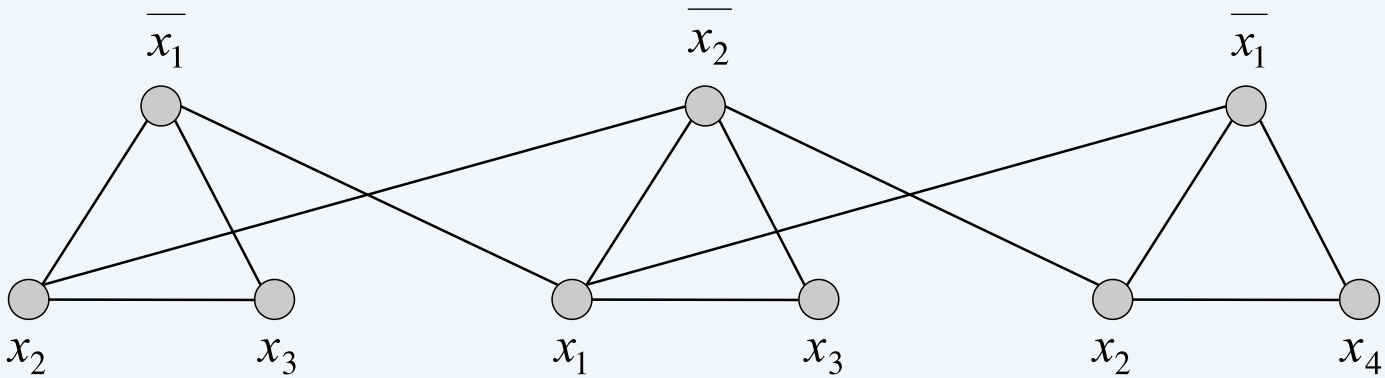
Lemma. G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable.

Pf. $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining variables consistently).
- Truth assignment is consistent and all clauses are satisfied.

Pf $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

G



k = 3

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

A proof that INDEPENDENT-SET is **NP**-complete

1. INDEPENDENT-SET $\in$ **NP**. We proved by showing that INDEPENDENT-SET has polynomial verifier.
2. 3-SAT is **NP**-complete. We assume we know it.
3. 3-SAT $\leq_P$ INDEPENDENT-SET. We provided a construction, and showed that it was polynomial.
4. Hence INDEPENDENT-SET is **NP**-complete.

Hitting Set Problem

- Consider a set $A = \{a_1, \ldots, a_n\}$ and a collection $B_1, B_2, \ldots, B_m$ of subsets of A (i.e., $B_i \subseteq A$ for each i).
- We say that a set $H \subseteq A$ is a *hitting set* for the collection $B_1, B_2, \ldots, B_m$ if H contains at least one element from each B_i - that is, if $H \cap B_i$ is not empty for each i (so H "hits" all the sets B_i).
- We now define the **Hitting Set Problem** as follows:

  We are given a set $A = \{a_1, \ldots, a_n\}$, a collection $B_1, B_2, \ldots, B_m$ of subsets of A, and a number k.
  We are asked: Is there a hitting set $H \subseteq A$ for $B_1, B_2, \ldots, B_m$ so that the size of H is at most k?

A proof that HITTING-SET is **NP**-complete

1. We need to show first that HITTING-SET $\in$ **NP**.
2. Assume that we know that VERTEX-COVER is **NP**-complete.
3. We now need to show that

$$\text{VERTEX-COVER} \leq_P \text{HITTING-SET}$$

NOT vice versa!

HITTING-SET $\in$ **NP**

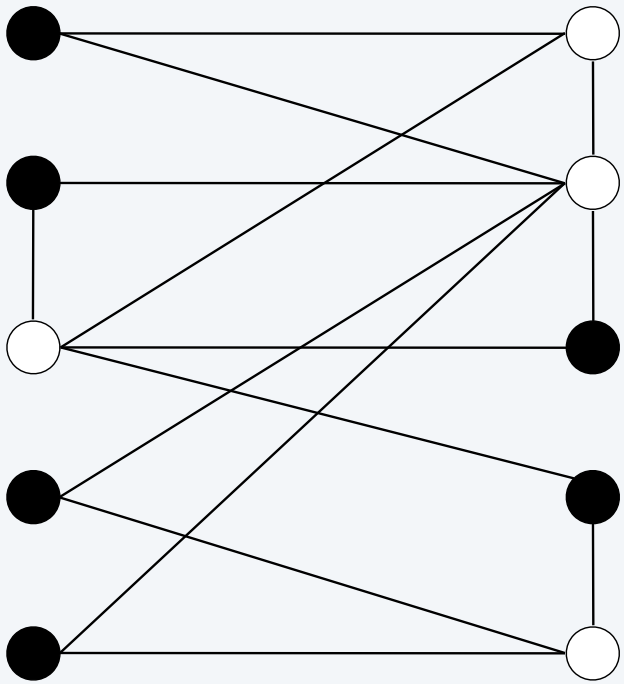
- We will show that an instance of the problem has **polynomial verifier**.
- Clearly given an instance of the problem, and a proposed set H, we can check in polynomial time whether H has size at most k (this can be done in $O(k)$), and whether some member of each set B_i belongs to H. Checking if $B_i \cap H \neq \emptyset$ is $O(|B_i||H|)$. Clearly $s|B_i||H| \leq n^2$ (as $B_i \subseteq A$ and $H \subseteq A$), so altogether we have $O(mn^2)$, i.e. polynomial time.
- Hence HITTING-SET $\in$ **NP**
- This part CANNOT be omitted!

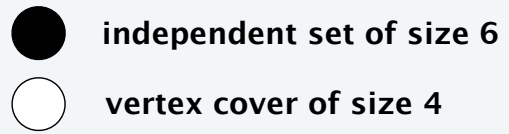
VERTEX-COVER in **NP**-complete

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?

Ex. Is there a vertex cover of size ≤ 3?





VERTEX-COVER $\leq_P$ HITTING-SET

- Assume that we know that VERTEX-COVER is **NP**-complete.
- HITTING-SET looks like a covering problem, since we are trying to choose at mist k objects subject to some constraints.
- He we will try to show VERTEX-COVER $\leq_P$ HITTING-SET.
- We begin with an instance of VERTEX-COVER, specified by a graph $G(V, E)$ and a number k.
- We must construct an equivalent instance of HITTING SET.
- In VERTEX-COVER, we are trying to choose at most k nodes to form a vertex cover.
- In HITTING-SET, we are trying to choose at most k elements to form a hitting set.
- This suggest that we define the set A in the HITTING-SET instance to be the V of nodes in the VERTEX-COVER instance. For each edge $e_i = (u_i, v_i) \in E$, we define a set $B_i = \{u_i, v_i\}$ in the HITTING-SET instance.
- The above construction is just some renaming so it is at most $O(|V|^2)$, so it is polynomial.

VERTEX-COVER $\leq_P$ HITTING-SET

- Now we claim that there is a hitting set of size an most k for this instance, if and only if the original graph has a vertex cover of size at most k.

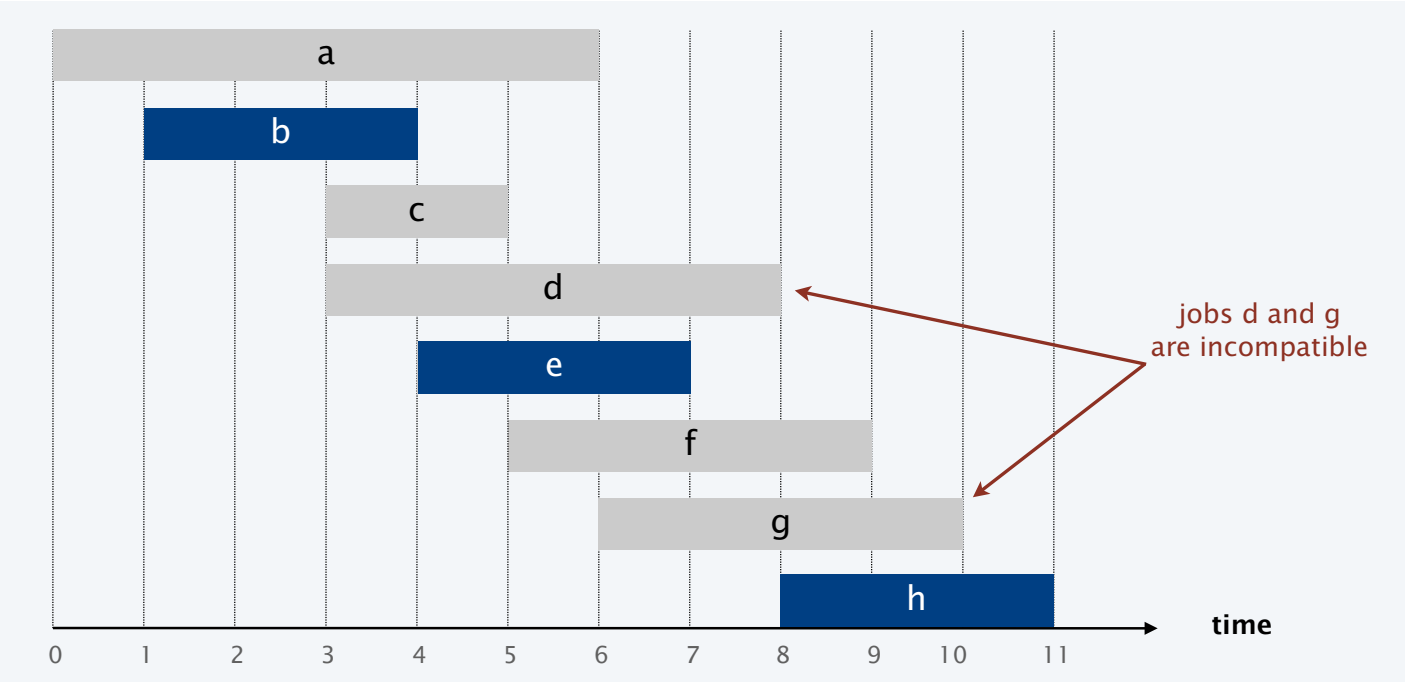
$\Rightarrow$ If we consider a hitting set H of size at most k as a subset of the nodes of G, we see that every set is "hit", and hence every edge has at least one end in H, H is a vertex cover of G.

$\Leftarrow$ If we consider a vertex cover C of G, and consider C as a subset of A, we see that each of the sets B_i is "hit" by C.

- Hence VERTEX-COVER $\leq_P$ HITTING-SET.
- Consequently **HITTING-SET is NP-complete.**

Consider Interval Scheduling problem from a class on Greedy Algorithms. It has a solution in $O(n \log n)$!

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



INTERVAL-SCHEDULING: For any given k, does there exists a subset of mutually compatible jobs $\geq k$?

NP-completeness and the class P

- INTERVAL-SCHEDULING: For any given k, does there exists a subset of mutually compatible jobs $\geq k$?
- INTERVAL-SCHEDULING has $O(n \log n)$ solution: we just find a maximum subset of mutually compatible jobs in $O(n \log n)$ using greedy algorithm from Lecture Notes 3, and then compare if the number of found jobs with k.
- Hence INTERVAL-SCHEDULING $\in$ **P**.
- Suppose I will give you an assignment question: Which of the below is true:
  1. INTERVAL-SCHEDULING $\leq_P$ HITTING-SET
  2. HITTING-SET $\leq_P$ INTERVAL-SCHEDULING

  What should be your answer?

(1) Is INTERVAL-SCHEDULING $\leq_P$ HITTING-SET? The answer is **YES**.

- Since HITTING-SET is **NP**-complete, then, by definition (see page 9 of this Lecture Notes), every $X \in \mathbf{NP}$ is *polynomially reducible* to HITTING-SET, i.e.

$$X \leq_P \text{HITTING-SET}.$$

- Since INTERVAL-SCHEDULING $\in \mathbf{P}$ and $\mathbf{P} \subseteq \mathbf{NP}$, then INTERVAL-SCHEDULING $\in \mathbf{NP}$.
- But this means

$$\text{INTERVAL-SCHEDULING} \leq_P \text{HITTING-SET!}$$

(2) Is HITTING-SET $\leq_P$ INTERVAL-SCHEDULING? The answer is **I DO NOT KNOW**. *Probably NOT.*

- We have a theorem:

Theorem

Suppose Y is **NP***-complete. Then $Y \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.*

- Since INTERVAL-SCHEDULING $\in \mathbf{P}$,
  if HITTING-SET $\leq_P$ INTERVAL-SCHEDULING
  then HITTING-SET $\in \mathbf{P}$!
- But HITTING-SET is **NP**-complete, so
  $$\text{HITTING-SET} \in \mathbf{P} \implies \mathbf{P} = \mathbf{NP}.$$
- Hence if you could prove
  $$\text{HITTING-SET} \leq_P \text{INTERVAL-SCHEDULING},$$
  you are one million of US dollars richer!