CSE 421
Introduction to Algorithms

Lecture 24: P, NP, NP-completeness

Polynomial time

YES / NO answer

Defn: Let **P** (polynomial-time) be the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

This is the class of decision problems whose solutions we have called "efficient".

Last time: Polynomial Time Reduction

Defn: We write $A \leq_P B$ iff there is an algorithm for A using a 'black box' (subroutine or method) that solves B that

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for B.

Theorem: If $A \leq_P B$ then a poly time algorithm for $B \Rightarrow$ poly time algorithm for A

Proof: Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!

Corollary: If you can prove there is no fast algorithm for A, then that proves there is no fast algorithm for B.

Intuition for "$A \leq_P B$": "B is at least as hard* as A" *up to polynomial-time slop.

Polynomial Time Reduction

Defn: We write $A \leq_P B$ iff there is an algorithm for A using a 'black box' (subroutine or method) that solves B that

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for B.

Theorem: If $A \leq_P B$ then $B \in \mathbf{P} \Rightarrow A \in \mathbf{P}$

Proof: Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!

Corollary: If $A \leq_P B$ then $A \notin \mathbf{P} \Rightarrow B \notin \mathbf{P}$.

Theorem: If $A \leq_P B$ and $B \leq_P C$ then $A \leq_P C$

Proof: Compose the reductions: Plug in "the algorithm for B that uses C" in place of B

Reminder: The terminology for reductions…

We read "$A \leq_P B$" as "A is polynomial-time **reducible** to B" or "A can be **reduced** to B in polynomial time"

- It means "we can solve A using at most a polynomial amount of work on top of solving B."
- But word reducible seems to go in the opposite direction of the $\leq$ sign.

Last time: Some reductions

Theorem: Independent-Set $\leq_P$ Clique

Theorem: Clique $\leq_P$ Independent-Set

Reminder: Reduction steps

4 steps for reducing (decision problem) A to problem B

1. Describe the reduction itself
   - i.e., the function f that converts the input x for A to the one for problem B.
2. Make sure the running time to compute f is polynomial
   - In lecture, we'll sometimes skip writing out this step.
3. Argue that if the correct answer to the instance x for A is **YES**, then the instance $f(x)$ we produced is a **YES** instance for B.
4. Argue that if the instance $f(x)$ we produced is a **YES** instance for B then the correct answer to the instance x for A is **YES**.

Another Reduction

Vertex-Cover:

Given a graph $G = (V, E)$ and an integer k

Is there a $W \subseteq V$ with $|W| \leq k$ such that every edge of G has an endpoint in W? (W is a vertex cover, a set of vertices that covers E.)

i.e., Is there a set of at most k vertices that touches all edges of G?

Claim: **Independent-Set $\leq_P$ Vertex-Cover**

Lemma: In a graph $G = (V, E)$ and $U \subseteq V$

U is an independent set $\Leftrightarrow$ $V - U$ is a vertex cover

Reduction Idea

Lemma: In a graph $G = (V, E)$ and $U \subseteq V$

U is an independent set $\Leftrightarrow$ $V - U$ is a vertex cover

Proof:

($\Rightarrow$) Let U be an independent set in G

Then for every edge $e \in E$,

U contains at most one endpoint of e

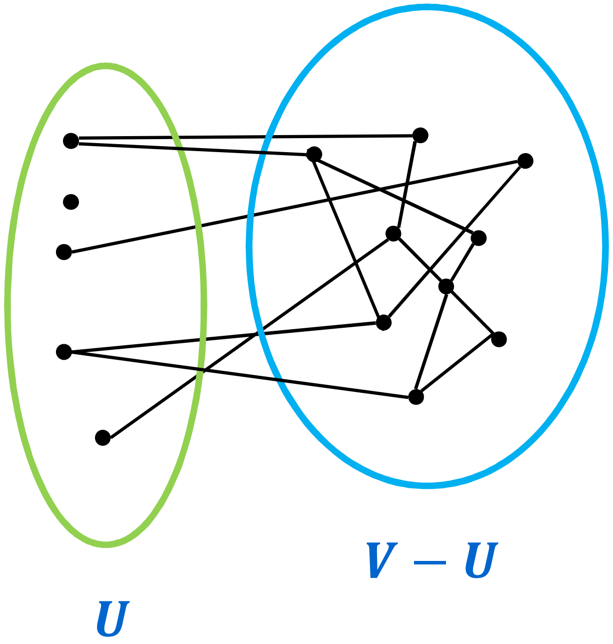
So, at least one endpoint of e must be in $V - U$

So, $V - U$ is a vertex cover

($\Leftarrow$) Let $W = V - U$ be a vertex cover of G

Then U does not contain both endpoints of any edge (else W would miss that edge)

So U is an independent set ■



Reduction for Independent-Set $\leq_P$ Vertex-Cover

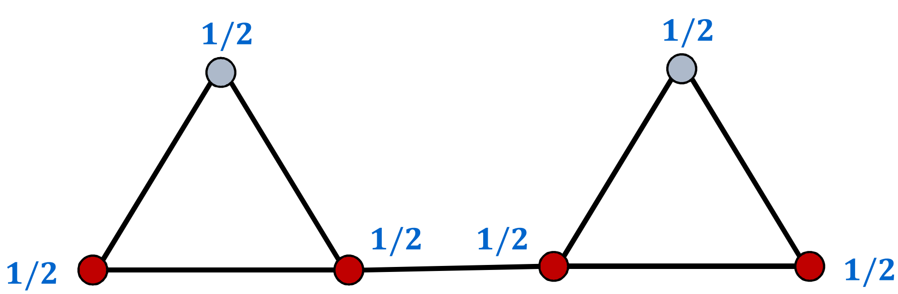
- Map (G, k) to $(G, n-k)$
  - Previous lemma proves correctness
- Clearly polynomial time
- Just as for Clique, we also can show
  - **Vertex-Cover $\leq_P$ Independent-Set**
    - Map (G, k) to $(G, n-k)$

Recall: Vertex-Cover as LP

Given: Undirected graph $G = (V, E)$

Q: Is there a set of at most k vertices touching all edges of G?

Doesn't work: To define a set we need $x_v = 0$ or $x_v = 1$



Natural Variables for LP:

x_v for each $v \in V$

Does this have a solution?

$\sum_v x_v \leq k$

$0 \leq x_v \leq 1$ for each node $v \in V$

$x_u + x_v \geq 1$ for each edge $\{u, v\} \in E$

LP minimum $= 3$

Vertex Cover minimum $= 4$

Integer-Programming, 01-Programming

Integer-Programming (ILP): Exactly like Linear Programming but with the extra constraint that the solutions must be integers. Decision version:

> **Given:** (integer) matrix $\boldsymbol{A}$ and (integer) vector $\boldsymbol{b}$
>
> Is there an integer solution to $\boldsymbol{Ax} \leq \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$?

01-Programming:

> **Given:** (integer) matrix $\boldsymbol{A}$ and (integer) vector $\boldsymbol{b}$
>
> Is there an solution to $\boldsymbol{Ax} \leq \boldsymbol{b}$ with $\boldsymbol{x} \in \{\boldsymbol{0}, \boldsymbol{1}\}$?

Then we have **Vertex-Cover** $\leq_P$ **01-Programming** $\leq_P$ **Integer-Programming**

Beyond P?

Independent-Set, **Clique**, **Vertex-Cover**, **01-Programming**, **Integer-Programming** and **3Color** are examples of natural and practically important problems for which we don't know any polynomial-time algorithms.

There are many others such as...

DecisionTSP:

Given a weighted graph G and an integer k,

Is there a tour that visits all vertices in G having total weight at most k?

and...

Satisfiability

- Boolean variables $x_1, \dots, x_n$
  - taking values in $\{0, 1\}$. 0=false, 1=true
- Literals
  - x_i or $\neg x_i$ for $i = 1, \dots, n$. ($\neg x_i$ also written as $\overline{x_i}$.)
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
- k-CNF formula
  - All clauses have exactly k variables

Satisfiability

CNF formula example:

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

Defn: If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable

- $(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$ is satisfiable: $x_1 = x_3 = 1$
- $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$ is not satisfiable.

3SAT: Given a CNF formula F with exactly 3 variables per clause, is F satisfiable?

Common property of these problems

- There is a special piece of information, a short certificate or proof, that allows you to **efficiently verify** (in polynomial-time) that the **YES** answer is correct. This certificate might be very hard to find.

  - **3Color**: the coloring.
  - **Independent-Set, Clique**: the set $\boldsymbol{U}$ of vertices
  - **Vertex-Cover**: the set $\boldsymbol{W}$ of vertices
  - **01-Programming**, **Integer-Programming**: the solution $\boldsymbol{x}$
  - **Decision-TSP**: the tour
  - **3SAT**: a truth assignment that makes the CNF formula $\boldsymbol{F}$ true.

The complexity class NP

N nondeterministic

NP consists of all decision problems where

- You can **verify** the **YES** answers efficiently (in polynomial time) given a short (polynomial-size) ***certificate***

and

- **No fake certificate** can fool your polynomial time verifier into saying **YES** for a **NO** instance

More precise definition of NP

A decision problem **A** is in **NP** iff there is

- a polynomial time procedure **VerifyA(.,.)** and
- a polynomial p

s.t.

- for every input x that is a **YES** for **A** there is a string t with $|t| \leq p(|x|)$ with **VerifyA**(x, t) = **YES**

and

- for every input x that is a **NO** for **A** there does not exist a string t with $|t| \leq p(|x|)$ with **VerifyA**(x, t) = **YES**

- A string t on which **VerifyA**(x, t) = **YES** is called a ***certificate*** for x or a ***proof*** that x is a **YES** input

Verifying the certificate is efficient

3Color: the coloring

- Check that each vertex has one of only 3 colors and check that the endpoints of every edge have different colors

Independent-Set, Clique: the set U of vertices

- Check that $|U| \geq k$ and either no (**IS**) or all (**Clique**) edges on present on U

Vertex-Cover: the set W of vertices

- Check that $|W| \leq k$ and W touches every edge.

01-Programming, **Integer-Programming**: the solution x

- Check type of x; plug in x and see that it satisfies all the inequalities.

Decision-TSP: the tour

- Check that tour touches each vertex and has total weight $\leq k$.

- **3-SAT**: a truth assignment α that makes the CNF formula F true.
  - Evaluate F on the truth assignment α.

Keys to showing that a problem is in NP

1. Must be decision probem (**YES**/**NO**)
2. For every given **YES** input, is there a certificate (i.e., a hint) that would help?
   - OK if some inputs don't need a certificate
3. For any given **NO** input, is there a fake certificate that would trick you?
4. You need a polynomial-time algorithm to be able to tell the difference.

Another NP problem

Sudoku:

- Is there a solution where this square has value 4?
- Certificate = full filled in table
  - Easy to check

9			5					
6	2		7			5		
		5				6		7
		6			4			
2				3			9	
	8						1	
4								8
			1	8		4		
7							2	

Fact: All **NP** problems could be solved efficiently by solving any of the problems on the previous slide efficiently or even by doing it for a general $n^2 \times n^2$ version of Sudoku!

Solving NP problems without hints

There is an obvious algorithm for all $\mathbf{NP}$ problems:

Brute force:

Try all possible certificates and check each one using the verifier to see if it works.

Even though the certificates are short, this is exponential time

- $\mathbf{2^n}$ truth assignments for $\boldsymbol{n}$ variables
- $\binom{\boldsymbol{n}}{\boldsymbol{k}}$ possible $\boldsymbol{k}$-element subsets of $\boldsymbol{n}$ vertices
- $\boldsymbol{n}!$ possible TSP tours of $\boldsymbol{n}$ vertices
- etc.

What We Know

- Every problem in $\mathbf{NP}$ is in exponential time
- Every problem in $\mathbf{P}$ is in $\mathbf{NP}$
  - You don't need a certificate for problems in $\mathbf{P}$ so just ignore any hint you are given
- Nobody knows if all problems in $\mathbf{NP}$ can be solved in polynomial time; i.e., does $\mathbf{P} = \mathbf{NP}$?
  - one of the most important open questions in all of science.
  - huge practical implications
- Most CS researchers believe that $\mathbf{P} \neq \mathbf{NP}$
  - $1M prize either way
  - but we don't have good ideas for how to prove this ...

NP-hardness & NP-completeness

Notion of hardness we **can** prove that is useful unless $\mathbf{P} = \mathbf{NP}$:

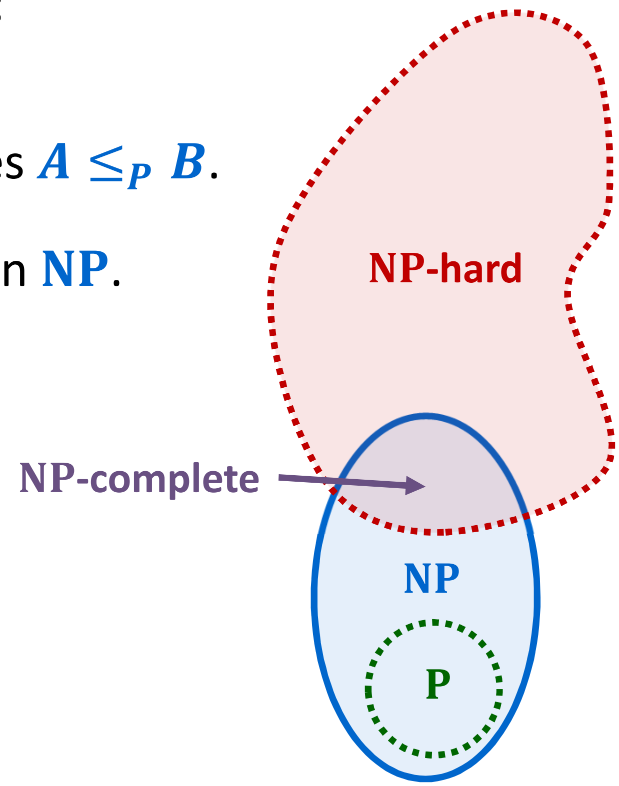
Defn: Problem B is $\mathbf{NP}$-hard iff **every** problem $A \in \mathbf{NP}$ satisfies $A \leq_P B$.

This means that B is at least as hard as every problem in $\mathbf{NP}$.

Defn: Problem B is $\mathbf{NP}$-complete iff

- $B \in \mathbf{NP}$ and
- B is $\mathbf{NP}$-hard.

This means that B is a hardest problem in $\mathbf{NP}$.



Not at all obvious that any $\mathbf{NP}$-complete problems exist!

Cook-Levin Theorem

Theorem [Cook 1971, Levin 1973]: **3SAT** is **NP**-complete

Proof: See CSE 431.

Corollary: If **3SAT** $\leq_P$ **B** then **B** is **NP**-hard.

Proof: Let **A** be an arbitrary language in **NP**.
Since **3SAT** is **NP**-hard we have **A** $\leq_P$ **3SAT.**

Then **A** $\leq_P$ **3SAT** and **3SAT** $\leq_P$ **B** imply that **A** $\leq_P$ **B**.

Therefore every language **A** in **NP** has **A** $\leq_P$ **B** so **B** is **NP**-hard.

Cook & Levin did the hard work.

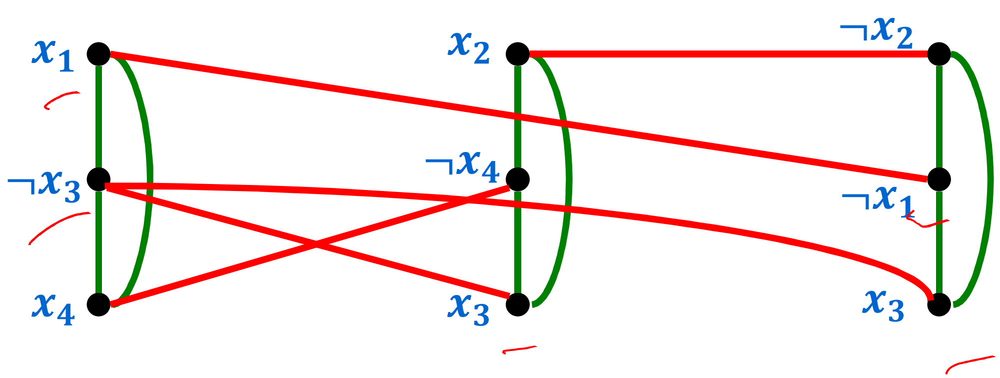
We only need to give one reduction to show that a problem is NP-hard!

Another NP-complete problem: 3SAT $\leq_P$ Independent-Set

1. The reduction:
    - Map 3CNF formula F to a graph G and integer k
    - Let m = # of clauses of F
    - Create a vertex in G for each literal occurrence in F
    - Join two vertices u, v in G by an edge iff
        - u and v correspond to literals in the same clause of F (green edges) or
        - u and v correspond to literals x and $\neg x$ (or vice versa) for some variable x (red edges).
    - Set $k = m$
2. Clearly polynomial-time computable

Another NP-complete problem: 3SAT $\leq_P$ Independent-Set

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



$m = 3$

G has both kinds of edges.
The color is just to show why the edges were included.

$k = m = 3$

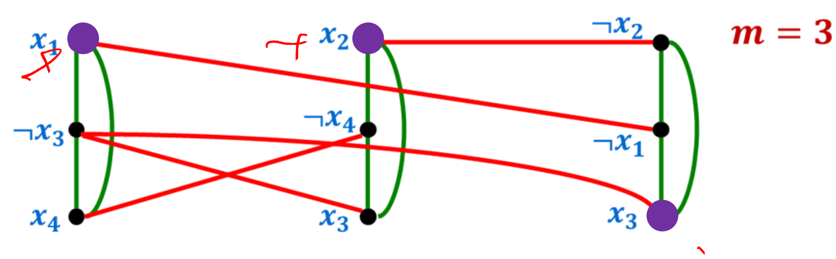
Correctness (⇒)

Suppose that F is satisfiable (**YES** for **3SAT**)

- Let α be a satisfying assignment; it satisfies at least one literal in each clause.
- Choose the set U in G to correspond to the **first satisfied literal in each clause**.
  - $|U| = m$
  - Since U has 1 vertex per clause, no green edges inside U.
  - A truth assignment never satisfies both x and $\neg x$, so no red edges inside U.
  - Therefore U is an independent set of size m

Therefore (G, m) is a **YES** for **Independent-Set**.

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



Satisfying assignment α:

$\alpha(x_1) = \alpha(x_2) = \alpha(x_3) = \alpha(x_4) = 1$

Set U marked in purple is independent.

Correctness (⇐)

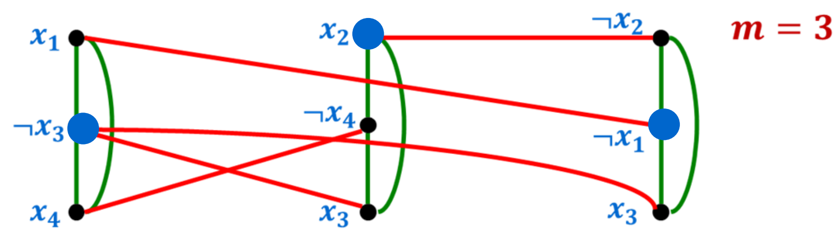
Suppose that G has an independent set of size m

((G, m) is a **YES** for **Independent-Set**)

- Let U be the independent set of size m;
- U must have one vertex per column (green edges)
- Because of red edges, U doesn't have vertex labels with conflicting literals.
- Set all literals labelling vertices in U to true
- This may not be a total assignment but just extend arbitrarily to a total assignment α.
    - This assignment satisfies F since it makes at least one literal per clause true.

Therefore F is satisfiable and a **YES** for **3SAT**.

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



Given independent set U of size m

Satisfying assignment α: Part defined by U:

$\alpha(x_1) = 0, \alpha(x_2) = 1, \alpha(x_3) = 0$

Set $\alpha(x_4) = 0$.

Many NP-complete problems

Since **3SAT** $\leq_P$ **Independent-Set**, **Independent-Set** is **NP**-hard.

We already showed that **Independent-Set** is in **NP**.

⇒ **Independent-Set** is **NP**-complete

Corollary: **Clique**, **Vertex-Cover**, **01-Programming**, and **Integer-Programming** are also **NP**-complete.

Proof: We already showed that all are in **NP**.

We also showed that **Independent-Set** polytime reduces to all of them.

Combining this with **3SAT** $\leq_P$ **Independent-Set** we get that all are **NP**-hard. ■