CMPSC 465
Data Structures and Algorithms
Fall 2024

Instructor: Debarati Das

December 6, 2024

NP and Computational Hardness

NP and Computational Hardness

Polynomial-time reduction (Kleinberg-Tardos, Section 8.1, 8.2)

Motivation

Which problem is harder?

- Problem A: it takes me a week to come up with an $O(n^2)$ algorithm
- Problem B: It's straightforward to design a brute-force algorithm with running time $O(2^n)$, but it's the best-known algorithm

Is Problem B really hard? How do we prove hardness?

Proving hardness

We can prove **lower bound** for some problems. For example: for sorting $\Omega(n \log n)$

For hard problems, can we get something like $\Omega(2^n)$?
Unfortunately, for most hard problems, we can't either find an $O(\mathrm{poly}(n))$ time algorithm or prove a lower bound like $\Omega(\exp(n))$

Instead, we classify hard computational problems. We prove they are "equivalent" in the sense that

- A polynomial-time algorithm for any one of them would imply there exist polynomial-time algorithms for all of them

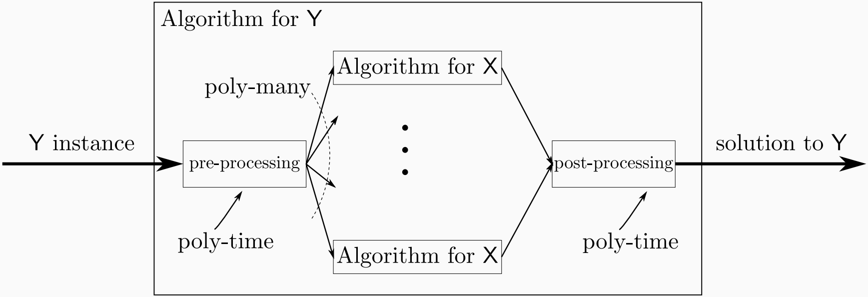
Tool: **polynomial-time** reduction

Polynomial-time reduction

Definition

A problem Y is **polynomial-time reducible** to a problem X if there exists an algorithm that solves any instance of Y making polynomially many elementary operations and polynomially many calls to a black-box solving X

Denote it by Y $\leq_P$ X



Consequences of polynomial time reductions

Lemma

Suppose $Y \leq_P X$. *If* X *can be solved in polynomial time, then* Y *can be solved in polynomial time*

Intuition: X is at least as hard as Y

Lemma

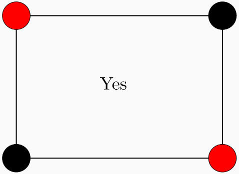
Suppose $Y \leq_P X$. *If* Y *cannot be solved in polynomial time, then* X *cannot be solved in polynomial time*

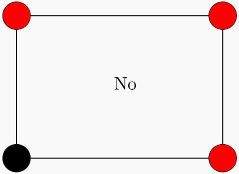
Independent set (of a graph)

Definition

A set of vertices is said to be **independent**, if no two of them are connected by an edge

Independent set?





The Maximum Independent Set Problem

The Maximum Independent Set Problem (Decision version)

Instance: a graph G, a number k

Objective: Decide if G contains an independent set of size k?

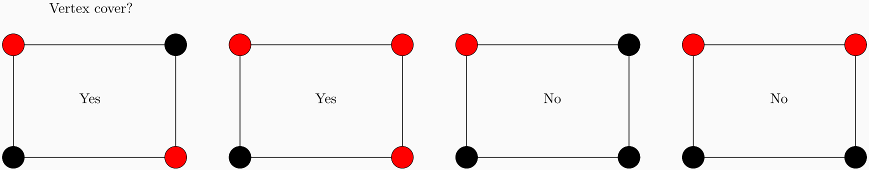
Optimization version: Find the maximum independent set

- Decision version $\leq_P$ optimization version
- Optimization version $\leq_P$ decision version (binary search)

Vertex Cover of a graph

Definition

A set of vertices is said to be a **vertex cover** if every edge has at least one end in it



The Minimum Vertex Cover Problem (Decision version)

Instance: a graph G, a number k

Objective: Decide if G contains a vertex cover of size k?

Independent Set and Vertex Cover (I)

Lemma

Let $G = (V, E)$ be a graph. Then S is an independent set if and only if its complement $V - S$ is a vertex cover

Proof.

- **"only if":** Suppose S is an independent set. Consider an arbitrary edge $e = (u, v)$. We know u, v cannot be both in S — one of them must be in $V - S$. So every edge has at least one end in $V - S$. So $V - S$ is a vertex cover
- **"if":** Suppose $V - S$ is a vertex cover. Consider any two vertices u, v in S. If u, v were joined by an edge, then neither of u, v would be in $V - S$, contradicting the assumption that $V - S$ is a vertex cover. So no two vertices in S are jointed by an edge. So S is an independent set $\square$

Independent Set vs Vertex Cover (II)

Theorem

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

Satisfiability

Recall Horn formulas are easy to solve

How about more general formulas: CNF (conjunction normal form)?

Definition

A **CNF formula** is a conjunction of clauses, where each clause is a disjunction of literals

Example: $(x_1 \vee x_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (\bar{x}_4 \vee x_7)$

Definition

A **k-CNF** is a CNF where each clause contains exactly k literals

The Satisfiability Problem (SAT)

Instance: A CNF Φ

Objective: Decide if Φ is satisfiable, i.e., is there an assignment so that Φ is true?

The k-Satisfiability Problem (k-SAT)

Instance: A k-CNF Φ

Objective: Decide if Φ is satisfiable

3-SAT and Independent Set

Theorem

3-SAT $\leq_P$ Independent Set

Proof. First consider an intuition for solving SAT:

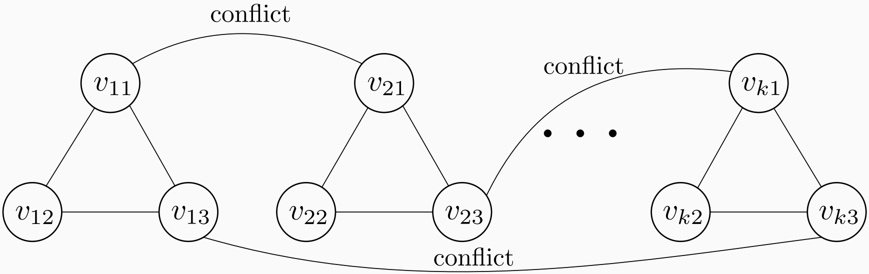
- pick one literal from each clause
- select an assignment that satisfies all selected literals
- make sure there's no conflict: Don't pick x from one clause and $\bar{x}$ from another

$$\Phi = (\overset{\substack{\text{bad}\\ \downarrow}}{\underset{\substack{\uparrow\\ \text{good}}}{x_1}} \vee x_2 \vee x_3) \wedge (\underset{\uparrow}{x_2} \vee \overset{\downarrow}{\bar{x}_3} \vee x_4) \wedge (\overset{\downarrow}{\underset{\uparrow}{x_3}} \vee \bar{x}_1 \vee x_5)$$

We encode a CNF as a graph, and encode an assignment as independent sets (to keep track of the conflicts)

Consider a 3-SAT instance with variables $x_1, \ldots, x_n$, and clauses $C_1, \ldots, C_k$

We build a graphs $G = (V, E)$ with $3k$ vertices, grouped into k triangles. Each triangle contains v_{i1}, v_{i2}, v_{i3} where v_{ij} corresponds to the j-th literal in C_i. Add edges for conflicts, i.e., x_j and $\bar{x}_j$:



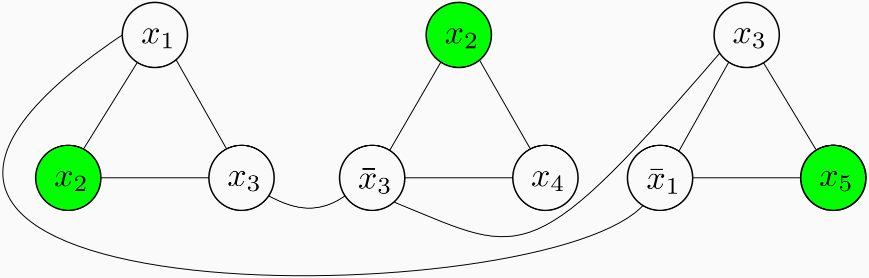
At most one vertex in each triangle can be in an independent set, so the size of an independent set cannot be larger than k

- If there exists a satisfying assignment, there exists a satisfied literal in each clause (triangle). Pick such a literal and include it into the independent set
  There is no conflicts. It's in fact an independent set
- If there exists an independent set S of size k, every triangle contains a vertex from S. We can choose an assignment so that all literals (vertices of S) are satisfied — there's no conflicts

So the 3-CNF has a satisfying assignment if and only if G has an independent set of size k $\square$

Example of the reduction

Consider $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_1 \vee x_5)$



Satisfying assignment: $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1$

NP and Computational Hardness

P, NP, and NP-completeness (Kleinberg-Tardos, Section 8.3, 8.4)

Problems and algorithms

We can encode the input (an instance) of any computational problem as a binary string

A **decision problem** X is the set of strings on which the answer is "yes"

An **algorithm** A for a decision problem receives an input string s and outputs $A(s) = \begin{cases} \text{yes} \\ \text{no} \end{cases}$

The algorithm A **solves** X if for all s, $A(s) = \text{yes}$ if and only if $s \in X$

The algorithm A has **polynomial running time** if there is a polynomial p s.t. for all s, A terminates on s in at most $O(p(|s|))$ steps

Computational class

P : the class of all problems for which there exists a polynomial-time algorithm

Checking vs solving

Definition

An algorithm B is an **efficient certifier** for a problem X if

- B is a polynomial-time algorithm that takes two inputs s, t, and
- there exists a polynomial p s.t. for all s, we have $s \in \mathrm{X}$ if and only if there exists a string t s.t. $|t| \leq p(|s|)$ and $B(s, t) = \text{yes}$

The string t is called a **certificate**

Example:

- 3-SAT: certificate: an assignment
  instance s: $(\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$
  certificate t: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$
- Independent set. certificate: a set of at least k vertices
  certifier: check if there's no edge joining them

We can use B to design an algorithm for X: use brute force to find a t.

But there might be exponentially many possible t's

The computational class NP

Computational class

NP : the class of all problems for which there exists an efficient certifier

It is easy to see: 3-SAT $\in$ **NP**

Lemma

P $\subseteq$ **NP**

Proof.

For any problem in **P** with algorithm A, we construct a certifier B that just returns $A(s)$ with empty certificate t $\square$

NP-completeness

Fundamental question in CS: is $\mathbf{P} = \mathbf{NP}$? i.e., does there exist a problem $X \in \mathbf{NP}$ but $X \notin \mathbf{P}$?

We don't know the answer, but we try to find the most difficult problems in **NP**:

Definition

A problem X is **NP-complete** if

- $X \in \mathbf{NP}$ and
- for all $Y \in \mathbf{NP}$, $Y \leq_P X$

Lemma

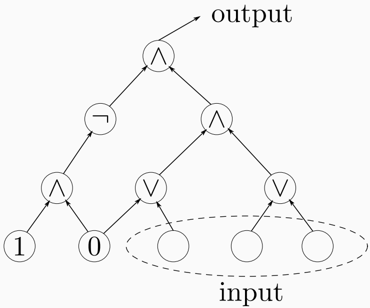
If an **NP***-complete problem can be solved in polynomial time, then* $\mathbf{P} = \mathbf{NP}$

Which problems are NP-complete?

A first **NP**-complete problem: Circuit Satisfiability

A circuit consists of

- inputs
- wires
- logical gates $\vee, \wedge, \neg$
- single output



The Circuit Satisfiability Problem (circuit-SAT)

Instance: A circuit C

Objective: Decide if C is satisfiable

The Cook-Levin Theorem

Theorem (Cook-Levin)

circuit-SAT is **NP***-complete*

Proof sketch. We need to reduce every problem $\mathrm{X} \in \mathbf{NP}$ to circuit-SAT
We use the fact that X has a polynomial-time certifier $B(\cdot, \cdot)$

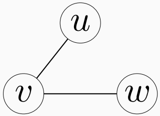
Main idea: any algorithm on inputs of fixed length can be simulated by a circuit, i.e., circuit outputs 1 if and only if algorithm outputs yes and if the algorithm takes polynomial time then the circuit has polynomial size

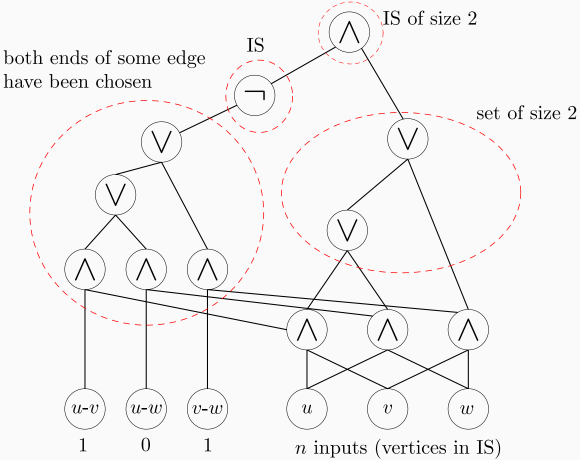
To decide if $s \in \mathrm{X}$, we check if there exists a string t of length $p(|s|)$ s.t. $B(s, t) = \text{yes}$
We transform $B(s, \cdot)$ into a circuit C_s with s "hardwired" and $p(|s|)$ inputs for possible t's

Ask if C_s is satisfiable. If yes, there exists such t so $s \in \mathrm{X}$.
If no, there's such t that $B(s, t) = \text{yes}$. So $s \notin \mathrm{X}$ $\square$

Example of such C_s

Decide if there's an IS of size 2





$\binom{n}{2}$ hardwired inputs (graph description)

Proving NP-completeness

Recipe for proving Y is **NP**-complete

Step 1: Prove $Y \in \mathbf{NP}$

Step 2: Choose an **NP**-complete problem X

Step 3: Prove $X \leq_P Y$

Observation

If X *is* $\mathbf{NP}$*-complete,* $Y \in \mathbf{NP}$*, and* $X \leq_P Y$*, then* Y *is* $\mathbf{NP}$*-complete*

Proof.

Let W be any problem in **NP**. Then $W \leq_P X \leq_P Y$ implies that $W \leq_P Y$. Therefore, Y is **NP**-complete $\square$

Theorem

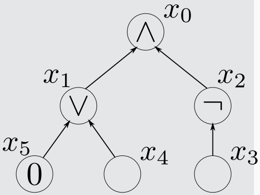
3-SAT is **NP***-complete*

Proof sketch.

We have seen that 3-SAT is in **NP**. Now we show circuit-SAT $\leq_P$ 3-SAT

Given a circuit, create a 3-SAT variable x_i for each circuit element i, e.g.,

- $x_2 = \bar{x}_3$: add 2 clauses, $(x_2 \vee x_3)$, $(\bar{x}_2 \vee \bar{x}_3)$
- $x_1 = x_5 \vee x_4$: add 3 clauses, $(x_1 \vee \bar{x}_4)$, $(x_1 \vee \bar{x}_5)$, $(\bar{x}_1 \vee x_4 \vee x_5)$
- $x_0 = x_1 \wedge x_2$: add 3 clauses, $(\bar{x}_0 \vee x_1)$, $(\bar{x}_0 \vee x_2)$, $(x_0 \vee \bar{x}_1 \vee \bar{x}_2)$
- hardwired input $x_5 = 0$: add clause $(\bar{x}_5)$
- output: $x_0 = 1$: add clause (x_0)



Turn clauses of length < 3 into clauses of length exactly 3 $\square$

Other NP-complete problems

From last lecture:

- Independent Set is **NP**-complete
- Vertex Cover is **NP**-complete

Other **NP**-complete problems:

- Hamilton cycle. Given $G = (V, E)$ undirected. Is there a simple cycle that contains every vertex in V?
  3-SAT $\leq_P$ Directed Hamiltonian Cycle $\leq_P$ Hamiltonian Cycle
- Travelling Salesman (TSP)
  Given a set of cities, distances $d(u, v)$, a number D, is there a tour of length $\leq D$?
  Hamiltonian Cycle $\leq_P$ TSP

and many more...

Want to learn more about this topic? Take CMPSC 464