Algorithm Design, Analysis & Complexity

Lecture 10 - $\mathcal{NP}$ Completeness & Computational Intractability

Koushik Pal

University of Toronto

July 27, 2021

Recap

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called **polynomial-time computable** if there exists a polynomial-time algorithm A that, given any input $x \in \{0,1\}^*$, produces as output $f(x)$.

Let X and Y be two problems. We say Y is **polynomial-time reducible** to X (or, X is at least as hard as Y w.r.t. polynomial time) if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in Y \iff f(x) \in X \quad \forall x \in \{0,1\}^*.$$

Notation: $Y \leq_p X$.

Fact

Suppose $Y \leq_p X$.

1. *If X is solvable in polynomial time, then so is Y.*
2. *If Y is not solvable in polynomial time, then neither is X.*

Recap

Definition ($\mathcal{P}$)

$$\mathcal{P} := \{X \mid \exists \text{ a polytime algorithm } A \text{ that solves } X\}.$$

Definition (Efficient Certifier)

B is an **efficient certifier** for a problem X if:

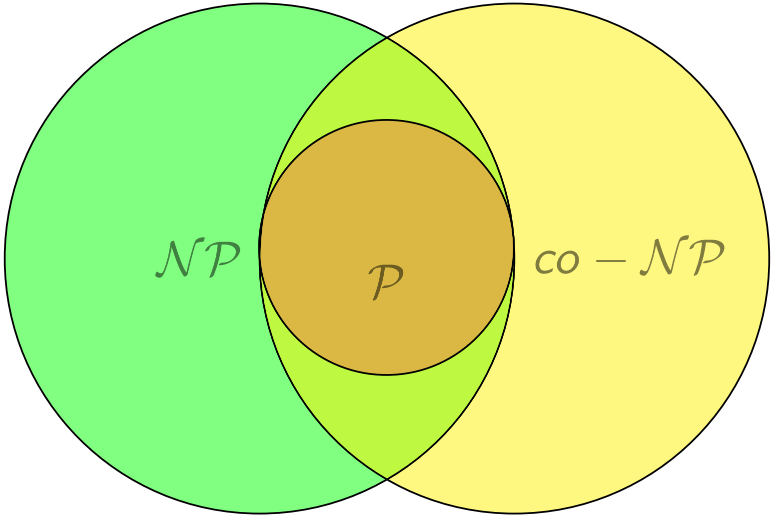
- B is a polytime algorithm that takes two inputs s and t.
- There is a polynomial function p so that for every string s, we have $s \in X \iff$ there exists a string t such that $|t| \leq p(|s|)$ and $B(s,t) =$ "yes".

(B says that an input $s \in X \iff$ there exists a proposed proof t that is not too long and that will convince that $s \in X$.)

Definition (Formal definition of $\mathcal{NP}$)

$$\mathcal{NP} := \{X \mid \exists \text{ an efficient certifier for decision problem } X\}.$$

Recap



Believed (but not known yet): $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{NP} \neq co\text{–}\mathcal{NP}$.

Unknown: $\mathcal{P} = \mathcal{NP} \bigcap co\text{–}\mathcal{NP}$?

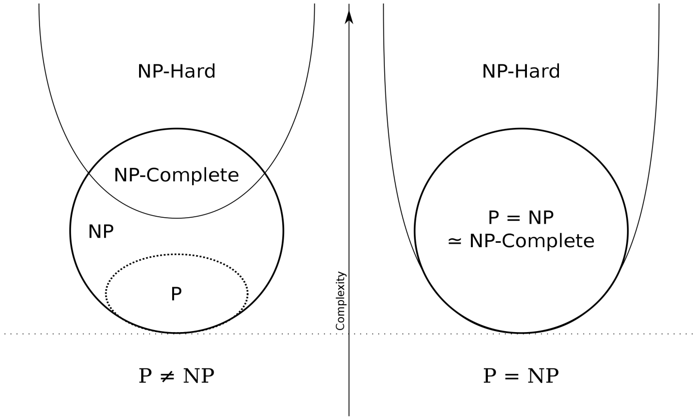
Recap

We use $\leq_p$ to identify "hardest problems" in $\mathcal{NP}$.

Definition

Decision problem D is called **$\mathcal{NP}$-complete** if

1. $D \in \mathcal{NP}$
2. D is **$\mathcal{NP}$-hard**: for all $D' \in \mathcal{NP}$, we have $D' \leq_p D$.



Theorem

If D is $\mathcal{NP}$-complete, then $D \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$.

How to prove $\mathcal{NP}$-completeness?

Lemma

To show D is $\mathcal{NP}$-hard, it is sufficient to find some $\mathcal{NP}$-hard D' and prove $D' \leq_p D$.

Proof.

$$
\begin{aligned}
D' \text{ is } \mathcal{NP}\text{-hard} \quad &\Longrightarrow \quad D'' \leq_p D' \text{ for all } D'' \in \mathcal{NP} \\
&\Longrightarrow \quad D'' \leq_p D \text{ for all } D'' \in \mathcal{NP} \quad (\text{since } D' \leq_p D) \\
&\Longrightarrow \quad D \text{ is } \mathcal{NP}\text{-hard.}
\end{aligned}
$$

$\square$

Corollary

To show D is $\mathcal{NP}$-complete, it is sufficient to show

- $D \in \mathcal{NP}$, *and*
- $D' \leq_p D$ *for some $\mathcal{NP}$-hard D'.*

Satisfiability

Example

Different versions of the Satisfiability problem:

SAT: Given a propositional formula φ, is there some setting of the variables that will make φ TRUE?

CNF-SAT: Given a propositional formula φ in Conjunctive Normal Form (CNF), is φ satisfiable? A formula φ in CNF looks like

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n,$$

where each clause $C_i = a_1 \vee a_2 \vee \cdots \vee a_\ell$ with each literal a_j being either a variable x_k or a negated variable $\neg x_k$, e.g.,

$$\varphi = (x_1 \vee \neg x_2) \wedge (x_3 \vee x_4 \vee \neg x_5 \vee \neg x_6) \wedge (x_1 \vee \neg x_3).$$

3-SAT: Given a propositional formula φ in CNF with each clause containing exactly 3 literals, is φ satisfiable?

CIRCUIT-SAT: Given a circuit built out of AND ($\wedge$), OR ($\vee$) and/or NOT ($\neg$) gates with a single output node, is there an assignment of values to the inputs that cause the output to take the value 1?

First example of $\mathcal{NP}$-completeness

Theorem (Cook-Levin)

CIRCUIT-SAT *is* $\mathcal{NP}$*-complete.*

First example of $\mathcal{NP}$-completeness

Proof Sketch.

- $\textsc{Circuit-Sat} \in \mathcal{NP}$: Given a particular assignment of the inputs, evaluate each connective one-by-one until you reach the final output node. This can be done in polynomial time. If the final output is 1, the circuit is satisfiable. Moreover, if φ is satisfiable, then there is some certificate that will make this verifier output "yes".
- $\textsc{Circuit-Sat}$ is $\mathcal{NP}$-hard: Let $D \in \mathcal{NP}$. Let $B(s,t)$ be a polytime verifier for D. This verifier can be implemented as a circuit with input gates representing the values of s and t. For any input s for D, we can hard code the value of s into this circuit in such a way that there is a value of the certificate for which the verifier outputs "yes" if and only if there is some setting of the input gates corresponding to t that makes the circuit output 1. It is possible to show that this transformation can be done in polytime (as a function of the size of s).

$\square$

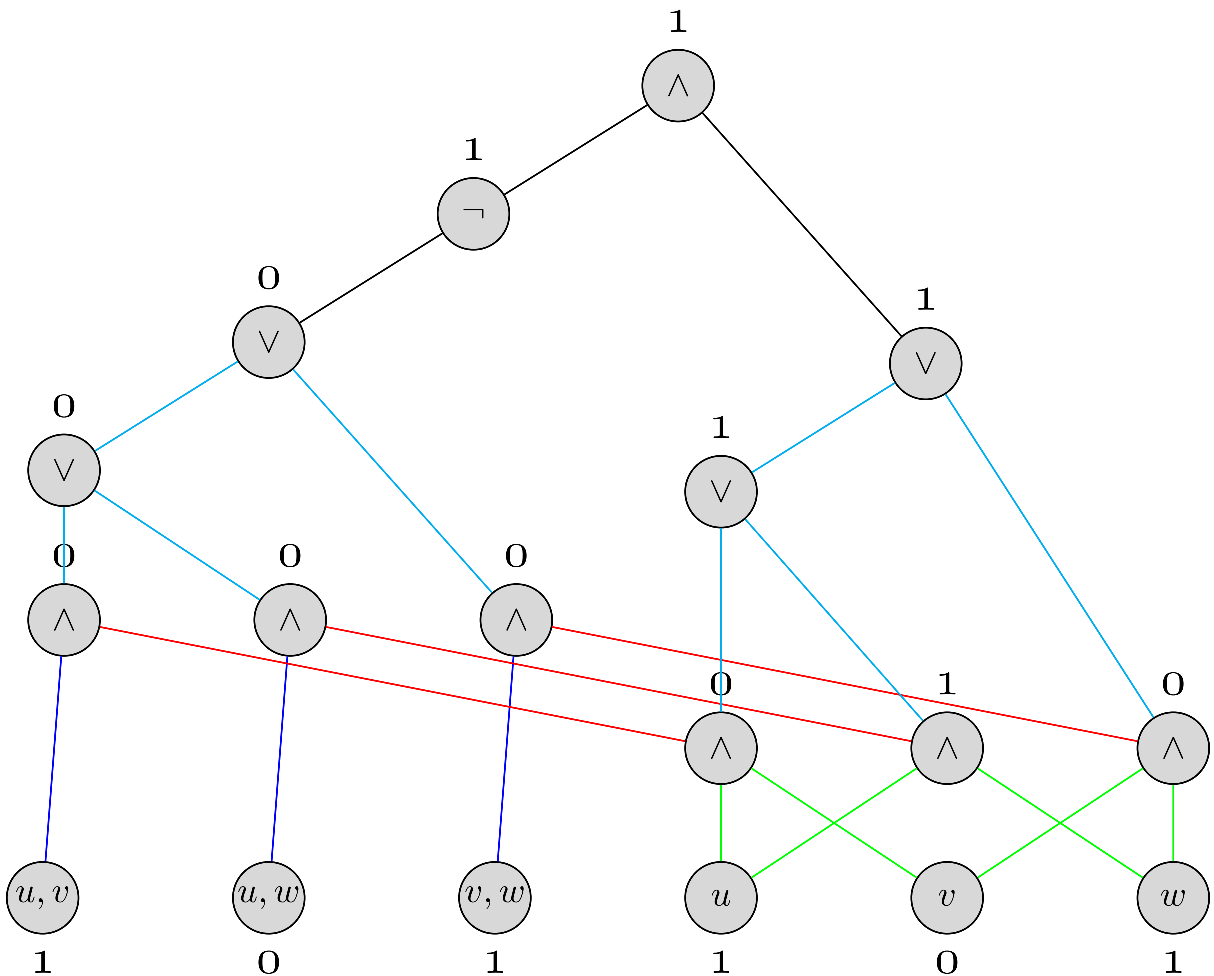
First example of $\mathcal{NP}$-completeness

Example (for Cook's Theorem)

Given a graph G, does it contain a two-node independent set?

First example of $\mathcal{NP}$-completeness

Assume $G = (V, E)$ with $V = \{u, v, w\}$ and $E = \{(u, v), (v, w)\}$.



SAT and CNF-SAT

Since every circuit made of AND, OR and/or NOT gates with one output node can be converted into a propositional formula with the variables corresponding to the input sources, and since every propositional formula can be converted into an equivalent CNF, it follows immediately that

Corollary

SAT *and* CNF-SAT *are both* $\mathcal{NP}$*-hard (and consequently,* $\mathcal{NP}$*-complete).*

3-SAT

Theorem

3-SAT is $\mathcal{NP}$-complete.

Proof.

Since 3-SAT is a special case of SAT, it follows that 3-SAT $\in \mathcal{NP}$.

We now show that CNF-SAT $\leq_p$ 3-SAT: given a CNF formula φ, construct a 3-CNF formula φ' such that

$$\varphi \text{ is satisfiable} \iff \varphi' \text{ is satisfiable.}$$

3-SAT

For each clause C of φ:

- if $C = (a_1)$, replace C with $(a_1 \vee a_1 \vee a_1)$
- if $C = (a_1 \vee a_2)$, replace C with $(a_1 \vee a_1 \vee a_2)$
- if $C = (a_1 \vee a_2 \vee a_3)$, leave C as is
- if $C = (a_1 \vee a_2 \vee \cdots \vee a_r)$ where $r > 3$, replace C with

$$\begin{array}{ll} & (a_1 \vee a_2 \vee z_1) \\ \wedge & (\neg z_1 \vee a_3 \vee z_2) \\ \wedge & (\neg z_2 \vee a_4 \vee z_3) \\ \wedge & \cdots \\ \wedge & (\neg z_{r-4} \vee a_{r-2} \vee z_{r-3}) \\ \wedge & (\neg z_{r-3} \vee a_{r-1} \vee a_r), \end{array}$$

where $z_1, \ldots, z_{r-3}$ are new variables (not in φ).

Clearly this transformation can be carried out in polytime : each clause of length r gets replaced with $\mathcal{O}(r)$ 3-clauses using $\mathcal{O}(r)$ new variables.

3-SAT

If φ is satisfiable, then there is an assignment of truth values to the variables of φ that makes at least one literal true in each clause of φ. This can be extended to include values for new variables of φ':

- trivial for $1-, 2-, 3-$clauses of φ
- for r-clauses of φ with $r > 3$, suppose a_i is true
  - if $i = 1$ or $i = 2$, set $z_1 = z_2 = \cdots = z_{r-3} = \text{FALSE}$
  - if $i = r-1$ or $i = r$, set $z_1 = z_2 = \cdots = z_{r-3} = \text{TRUE}$
  - if $2 < i < r-1$, set $z_1 = z_2 = \cdots = z_{i-2} = \text{TRUE}$ and $z_{i-1} = z_i = \cdots = z_{r-3} = \text{FALSE}$.

Verify that these assignments satisfy φ'.

3-Sat

Conversely, if φ' is satisfiable, then let z_i be the first new variable set to $\textsc{False}$ (so either $i = 1$ or $z_1 = z_2 = \cdots = z_{i-1} = \textsc{True}$ or all the z_i are $\textsc{True}$).

- if $i = 1$, the clause $(a_1 \vee a_2 \vee z_1)$ can only be satisfied by setting $a_1 = \textsc{True}$ or $a_2 = \textsc{True}$.
- if $i > 1$, the clause $(\neg z_{i-1} \vee a_{i+1} \vee z_i)$ can only be satisfied by setting $a_{i+1} = \textsc{True}$.
- if all the z_i are $\textsc{True}$, the clause $(\neg z_{r-3} \vee a_{r-1} \vee a_r)$ can only be satisfied by setting $a_{r-1} = \textsc{True}$ or $a_r = \textsc{True}$.

In all cases, $(a_1 \vee a_2 \vee \cdots \vee a_r)$ is also satisfied.

The cases of $1-, 2-, 3-$clauses are trivial again.

Thus, $\textsc{Cnf-Sat} \leq_p \text{3-}\textsc{Sat}$.

Since $\textsc{Cnf-Sat}$ is $\mathcal{NP}$-hard and $3\text{-}\textsc{Sat} \in \mathcal{NP}$, it follows that 3-Sat is $\mathcal{NP}$-complete. $\square$

SubsetSum

Definition (SubsetSum)

Given a finite set S of positive integers, and a positive integer target t, is there some subset S' of S whose sum is exactly t?

Theorem

SubsetSum *is $\mathcal{NP}$-complete.*

$\textsc{SubsetSum}$

Proof.

$\textsc{SubsetSum} \in \mathcal{NP}$ because it takes polytime to verify the certificate represents a subset of S whose sum is t.

We now show that 3-$\textsc{Sat} \leq_p \textsc{SubsetSum}$: given a 3-CNF formula φ, construct a set S of integers and an integer t such that

$$\varphi \text{ is satisfiable } \iff \exists S' \subseteq S \text{ whose sum is } t.$$

SubsetSum

Given $\varphi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_r \vee b_r \vee c_r)$, where $a_i, b_i, c_i \in \{x_1, \neg x_1, \ldots, x_s, \neg x_s\}$, construct S as follows:

- for $j = 1$ to s:

  number x_j = 1 followed by $s-j$ 0s followed by r digits where k^{th} next digit equals 1 if x_j appears in clause C_k, 0 otherwise.

  number $\neg x_j$ = 1 followed by $s-j$ 0s followed by r digits where k^{th} next digit equals 1 if $\neg x_j$ appears in clause C_k, 0 otherwise.

- for $j = 1$ to r:

  number C_j = 1 followed by $r-j$ 0s and

  number D_j = 2 followed by $r-j$ 0s.

- target $t = s$ 1s followed by r 4s.

Clearly, this can be constructed in polytime.

SubsetSum

Example

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

$$\begin{aligned}
x_1 &= 1000110 \\
\neg x_1 &= 1000000 \\
x_2 &= 100010 \\
\neg x_2 &= 100101 \\
x_3 &= 10000 \\
\neg x_3 &= 10011 \\
x_4 &= 1001 \\
\neg x_4 &= 1100 \\
D_1 &= 200 \\
C_1 &= 100 \\
D_2 &= 20 \\
C_2 &= 10 \\
D_3 &= 2 \\
C_3 &= 1 \\
t &= 1111444
\end{aligned}$$

SubsetSum

If φ is satisfiable, then there is a setting of variables such that each clause of φ contains at least one true literal.

Let $S' = \{$numbers that correspond to true literals$\}$.

By construction, $\sum_{x \in S'} x = s$ 1s followed by r digits, each of which is either 1, 2 or 3 (because each clause contains at least one true literal).

This means it is possible to add suitable numbers from $\{C_1, D_1, \ldots, C_r, D_r\}$ so that the last r digits of the sum are equal to 4, i.e., there is a subset S'' of S such that $\sum_{x \in S''} x = t$.

SubsetSum

Conversely, if there is a subset S' of S such that $\sum_{x \in S'} x = t$, then S' must contain exactly one of $\{x_j, \neg x_j\}$ for $j = 1$ to s.

Then φ is satisfied by setting each variable according to the numbers in S': for each clause j, the corresponding digit in the target is equal to 4, but the numbers C_j and D_j together only add up to 3 in that digit. This means that the selection of numbers in S' must include some literal with a 1 in that digit, i.e., clause C_j contains at least one true literal.

Since 3-Sat is $\mathcal{NP}$-hard and SubsetSum $\in \mathcal{NP}$, it follows that SubsetSum is $\mathcal{NP}$-complete.

$\square$

Categories of $\mathcal{NP}$-complete problems

- Packing Problems
  - IndependentSet
  - SetPacking
- Covering Problems
  - VertexCover
  - SetCover
- Partitioning Problems
  - 3-DimensionalMatching
  - GraphColoring
- Sequencing Problems
  - HamPath, UHamPath
  - HamCycle, UHamCycle
  - TravelingSalesman
- Numerical Problems
  - SubsetSum
  - Partition
  - Knapsack
  - IntegerProgramming
- Constraint Satisfaction Problems
  - Circuit-Sat, Sat
  - Cnf-Sat, 3-Sat

Class Diagram