NP and NP Completeness

Lecture 24

April 24, 2025

Part I

Efficient Computation: P and NP

What is Efficiency?

Last lecture, we discussed what problems can't have *any* algorithm—what changes if we care about *efficient* algorithms?

Definition

We say that a language L is in P if there exists an algorithm M that decides L, where for some constant c, $M(x)$ runs in time $O(|x|^c)$.

(Informally: P is the set of languages with polynomial-time algorithms.)

Why do we allow for *any* polynomial run time?

- Makes it simpler to describe algorithms.
- Polynomials have helpful closure properties: if $p(n)$ and $q(n)$ are polynomials, so are $p(n) + q(n)$, $p(n) \cdot q(n)$, and $p(q(n))$.
- We are interested in finding problems that *can't* be solved efficiently, so having a lax definition is more meaningful!

NP: Efficient Verification

Definition

We say that a language L is in NP if there is a polynomial $p(\cdot)$ and a machine M (running in time $O(|x|^c)$) such that:

- For every $x \in L$, there is a $w \in \{0,1\}^{p(|x|)}$ such that $M(x,w)$ accepts.
- For every $x \notin L$ and every $w \in \{0,1\}^{p(|x|)}$, $M(x,w)$ rejects.

Intuitively, L is in NP if it is easy to verify a proof that $x \in L$.

Examples we've already seen:

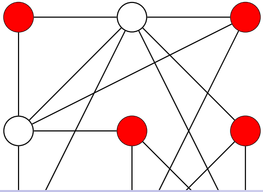
- Independent Set: w describes an IS of size k.
- Clique: w describes a clique of size k.

Why NP?

NP captures "most" problems we run into in the wild.

(Notable exception: the halting problem is *not* in ***NP***!)

We *think* that not all problems in ***NP*** can be solved efficiently: verifying answers seems easier than coming up with them!



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

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

P Versus NP

By definition, we have that $\boldsymbol{P} \subseteq \boldsymbol{NP}$.

Major open question: does $\boldsymbol{P} = \boldsymbol{NP}$?

- "Most" computer scientists conjecture no, but so far we can only prove that certain proof techniques aren't enough to show this!
- For 374, we will assume $\boldsymbol{P} \neq \boldsymbol{NP}$ unless otherwise stated, so *some* problem in $\boldsymbol{NP}$ cannot be solved efficiently.

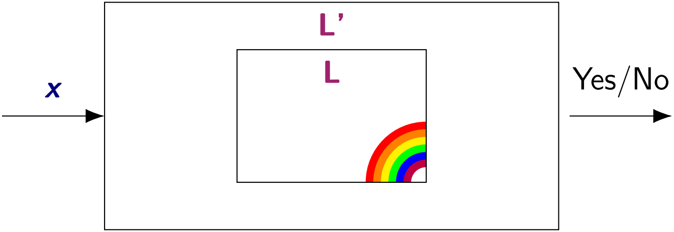
Can we come up with a *specific* language that isn't in $\boldsymbol{P}$?

Idea: if we can reduce *every* language in $\boldsymbol{NP}$ to some specific language $\boldsymbol{L}$, then we know $\boldsymbol{L}$ in particular is not in $\boldsymbol{P}$!

NP-Hard Languages

Definition

We say that L is NP-Hard if for every $L' \in NP$, there is a *polynomial-time* reduction from L' to L.



Requirement: as long as "magic box" for L runs in polynomial time, so does the "box" we build for L'.

NP-Hard Example

Claim

$L_{HNI} = \{\langle M \rangle \mid M$ *halts given no input*$\}$ *is* NP*-Hard.*

Let L be a language in NP, with M as the machine that verifies solutions and $p(\cdot)$ the polynomial such that $w \in \{0,1\}^{p(|x|)}$.

```
from magic import TestHNI
DecideL(x):
    Construct a program (machine) P() that:
        For each w ∈ {0,1}^p(|x|), runs M(x,w)
        If any iteration accepts, halt; else infinite loop
    return TestHNI(⟨P⟩)
```

Better question: is there an NP-hard problem *that is also in* NP? (We call such problems NP-complete.)

Part II

SAT

Boolean Satisfiablility (SAT)

Consider a boolean formula using AND, OR, and NOT:

$$((a \vee b \vee \overline{c}) \wedge \overline{(b \vee c)}) \vee d$$

Is there an assignment of True/False to a, b, c, and d such that this formula evaluates to True?

What about $a \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b})$?

Definition

Let $SAT = \{\varphi \mid \varphi \text{ is satisfiable}\}$

Note $SAT \in NP$: we can take w to be an assignment of True/False to each variable!

Using SAT

SAT turns out to be very powerful for modeling other problems!

Example: does $G = (V, E)$ have a *path* that visits every vertex?

The Cook-Levin Theorem

Theorem (Cook-Levin)

SAT *is* ***NP****-complete.*

Turns out all the formulas Cook-Levin constructs are all in "Conjunctive Normal Form":

- Formula is the AND of many clauses.
- Each clause is the OR of many variables/negations of variables.
- Example: $(a \lor \overline{b} \lor c) \land (b \lor d) \land (\overline{a} \lor b \lor c \lor \overline{d})$

Theorem (Cook-Levin, stronger version)

$\textbf{CNF-SAT} = \{\varphi \mid \varphi \text{ is satisfiable and in CNF}\}$ *is* ***NP****-complete.*

This means that (assuming $P \neq NP$) there is no polynomial time algorithm for ***SAT*** nor ***CNF-SAT***!

Part III

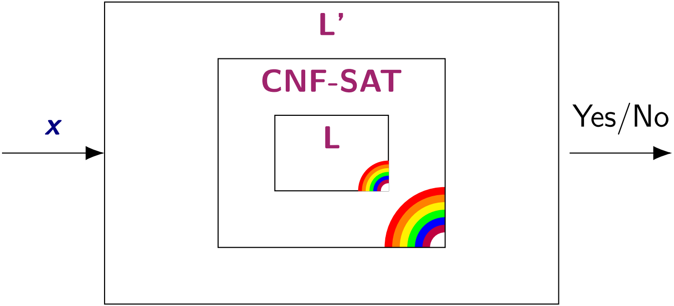
3SAT

Using Reductions

How do we prove that more problems are ***NP***-complete?

(Without redoing Cook-Levin...)

Idea: If ***CNF-SAT*** reduces to L in polynomial time, then L is ***NP***-hard! (So ***NP***-complete as long as $L \in NP$)



Common form of reduction: give function f (that can be computed

CNF-SAT to 3SAT I

A boolean formula is 3CNF if it is CNF and each clause has three literals. Let $\mathbf{3SAT}$ be the language of satisfiable 3CNF formulas.

Claim

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

$\mathbf{3SAT}$ is in $\mathbf{NP}$: $\mathbf{w}$ is a satisfying assignment to the variables.

Given a CNF formula, want to make each clause have 3 literals.

- How do we fix clauses with too few? (eg $(a \lor \overline{b})$)
- How do we fix clauses with too many? (eg $(a \lor \overline{b} \lor c \lor \overline{d})$)

CNF-SAT to 3SAT II

Claim

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

Given a CNF formula φ, create $f(\varphi)$ by for every $C \in \varphi$:

- If $C = (\ell_1)$ has one literal, include clauses $(\ell_1 \vee x_C \vee y_C)$, $(\ell_1 \vee \overline{x_C} \vee y_C)$, $(\ell_1 \vee x_C \vee \overline{y_C})$, and $(\ell_1 \vee \overline{x_C} \vee \overline{y_C})$ in $f(\varphi)$, where x_C and y_C are new variables.
- If $C = (\ell_1 \vee \ell_2)$ has two literals, include clauses $(\ell_1 \vee \ell_2 \vee x_C)$ and $(\ell_1 \vee \ell_2 \vee \overline{x_C})$ in $f(\varphi)$, where x_C is a new variable.
- If C has three literals, include C in $f(\varphi)$
- If $C = (\ell_1 \vee \ldots \vee \ell_k)$ has $k \geq 4$ literals, include clauses $(\ell_1 \vee \ell_2 \vee x_{C1})$, $(\overline{x_{C1}} \vee \ell_3 \vee x_{C2})$, ..., $(\overline{x_{C(k-3)}} \vee \ell_{k-1} \vee \ell_k)$ in $f(\varphi)$, where $(x_{C1}, \ldots, x_{C(k-3)})$ are new variables.

CNF-SAT to 3SAT III

Claim

$3SAT$ is NP-complete.

Our construction of f clearly runs in polynomial time. (In fact, quadratic.)

Exercise: formally prove that φ is satisfiable if and only if $f(\varphi)$ is.

- If direction: given a satisfiable assignment for $f(\varphi)$, find a satisfiable assignment for φ
- Only if direction: given a satisfiable assignment for φ, find a satisfiable assignment for $f(\varphi)$

Takeaway Points

Definitions of P and NP.

- If $L \in P$, we can efficiently decide if $x \in L$.
- If $L \in NP$, we can efficiently verify proofs that $x \in L$.
- We will assume that $P \neq NP$.

NP-hardness and NP-completeness

- A problem is NP-hard if *every* problem in NP reduces to it. If it is also in NP itself, we call it NP-complete.
- If you can reduce an NP-hard problem to L in polynomial time, L is also NP-hard.

Known NP-complete languages

- SAT
- CNF-SAT
- $3SAT$