CS 331, Fall 2025

Lecture 22 (11/17)

Today:
- Complexity theory
- Reductions
- P vs. NP
- NP-hardness

Complexity theory (Part VIII, Section 1)

In this class, we've seen many poly(n) algos (n = input size)

- All-pairs shortest paths
- Maxflow / mincut
- Linear programming

But many have <u>no known</u> poly(n) algo:

- Maxcut
- (Integer) vertex cover
- Subset sum * with exp(n)-sized weights

This is not for lack of trying!

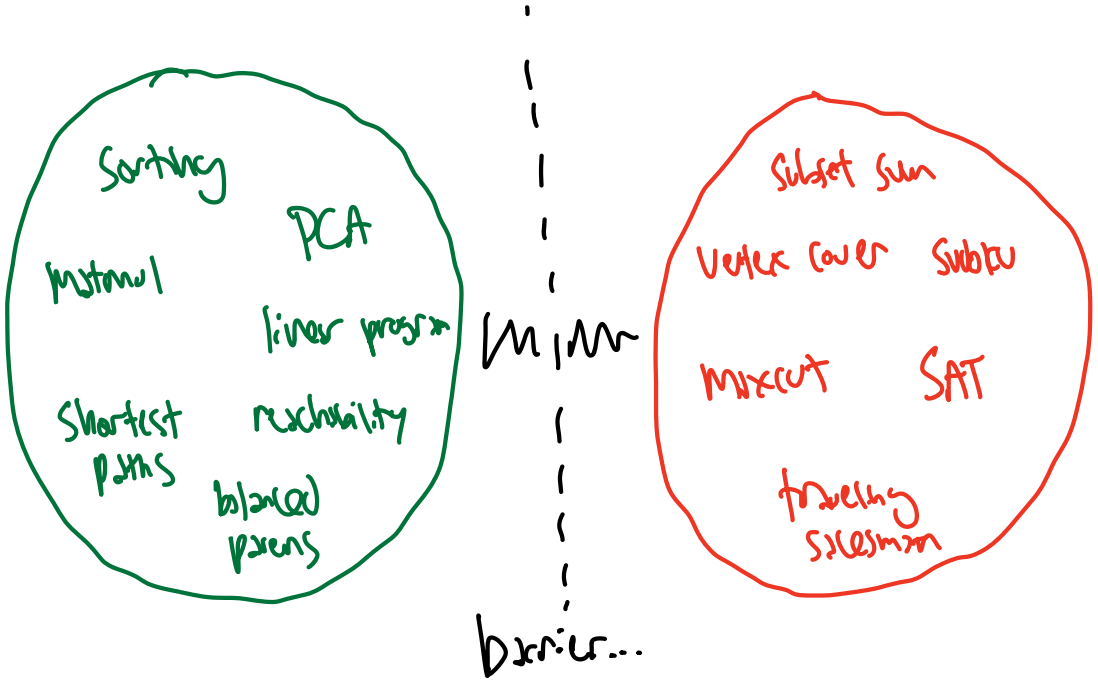
- Subset sum with poly(n)-sized weights ← DP
- Vertex cover with fractional values ← Liner program
- 0.878-approx. for maxcut

Still, some problems evade faster algos...

Complexity theory: Why are some problems easy while others are hard?

eg. poly(n)-time

Can we pinpoint common source of hardness?



Reductions (Part VIII, Section 2)

Detour to key concept: to formalize "barrier"

Consider two problems A and B

We write $A \leq_p B$ if:

$\exists$ algo to solve A that uses

- poly(n) calls to subroutine for B w/ poly(n) sized inputs
- poly(n) extra time

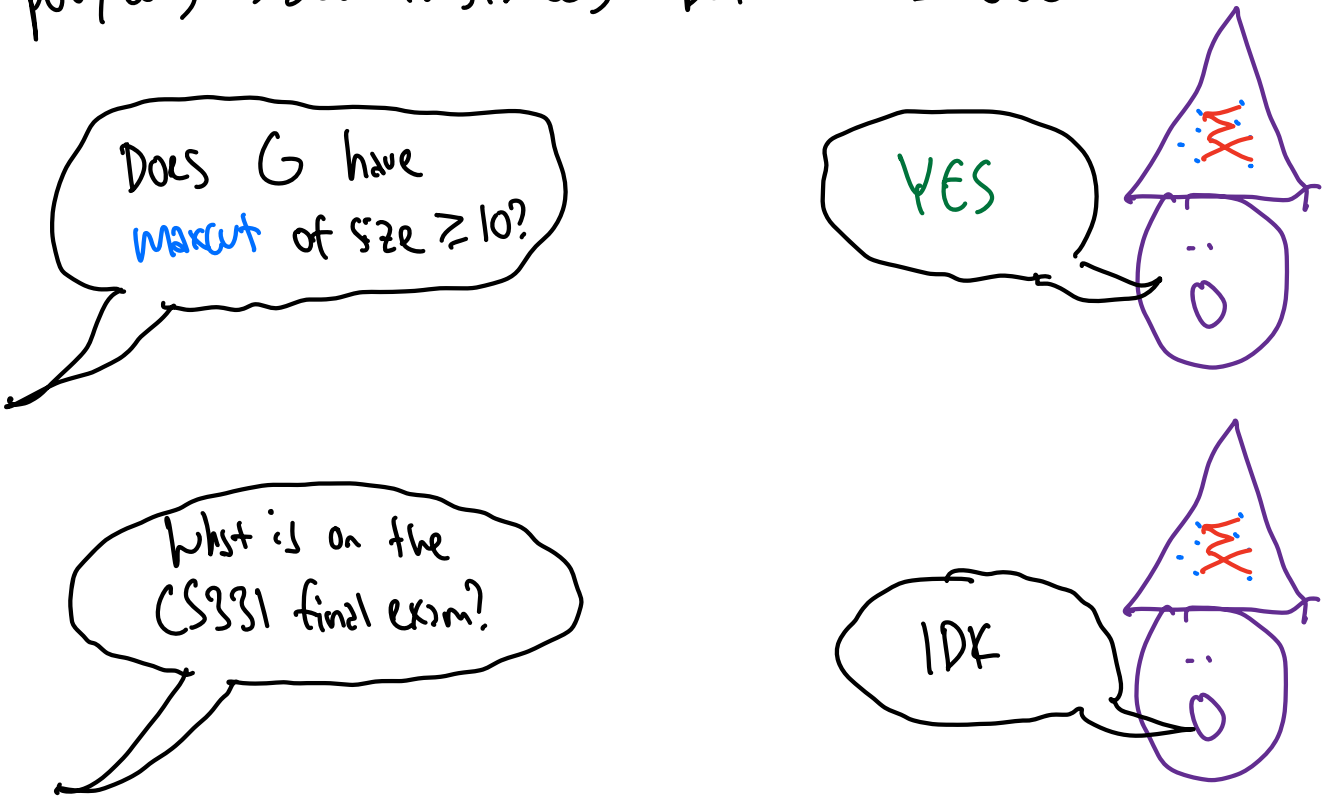
Algo For $A(x)$: // $n = |x|$

a ← ...
b ← ...
c ← Algo For $B(\ldots)$
Return True/False

} poly(n) total

Helpful viewpoint: oracles 

Imagine an oracle who can solve B "for free" on poly(n)-sized instances but nothing else



$A \leq_P B$: Solve A using poly(n) calls to B + extra time 

Fact: If B can be solved in poly(n) time + $A \leq_P B$

... then A can be solved in poly(n) time

e.g. We know  (thanks Edmonds-Karp etc!)

max-flow oracle
in poly(n) time

+ min-cut
disjoint paths
bipartite matching
tournament winning
...

Also poly-time!

$\leq_P$

max-flow

"bottleneck" problem

Some nitty gritty details...

1) Mainly focus on decision problems on binary inputs

True/False answers

$\{0,1\}^n$

Loses little generality:

- Convert optimization to decision

Max-flow OPT $(G) \rightarrow$ Max-flow DECIDE (G, F)

"is the max-flow value $\geq F$"

Binary search on F

- Convert input to binary

  Usually, some canonical encoding $\langle \text{input} \rangle \in \{0,1\}^n$

  $\langle V, E, w \rangle$ = binary (| m | n | head, | tail, | w_1 | ... |)

  $\langle V, E, w \rangle$: graph

We write L is a "language" (decision problem)

Each $x \in \{0,1\}^n$ is $\in L$ or not

e.g. 00010 → True ($x \in L$)

10101 → False ($x \notin L$)

2) Cook vs. Karp reductions

$$A \leq_p B$$

```
Cook Algo for A(x):
  a ∈ ...
  b ∈ Algo for B(x')
  c ∈ ...
  d ∈ Algo for B(x'')
  ...
```

Cook reduction

"anything goes"

OK for this class!

```
Karp Algo for A(x):
  a ∈ ...
  b ∈ ...
  ...
  Return Algo for B(x')
```

Karp reduction

"Call once, return immediately"

believed suffices for most problems

P vs. NP (Part VIII, Section 3.1)

The two most famous complexity classes.

P: All languages L that are decided (is $x \in \{0,1\}^n$ a member of L?) by a poly(n)-time algo.

e.g. reachability, APSP, matmul, ...

NP: All languages L that are decided by a poly(n)-non-deterministic-time algo.

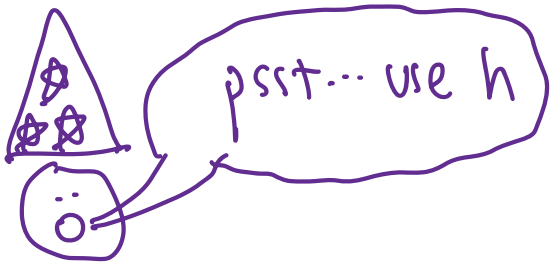
Non-determinism: poly(n)-sized "hint" h

$A(x,h)$: if $x \in L$, $\exists\ h$ s.t. $A(x,h) =$ True (can be hinted)

$x \notin L$, $\nexists\ h$ s.t. $A(x,h) =$ True (can't be tricked)

A ~ non-deterministic algo for L

Interpretation 1: advice



"NP oracle"

good at proving

? at refuting

You can be convinced of $x \in L$

if you talk to the oracle.

But no adversary can trick you into accepting $x \notin L$.

Interpretation 2: foresight

Interpret definition of $L \in NP$ as:

$$x \in L \iff \exists\, h \in \{0,1\}^{poly(n)} \text{ s.t. } A(x,h) = True$$

"best possible" — random coin flips

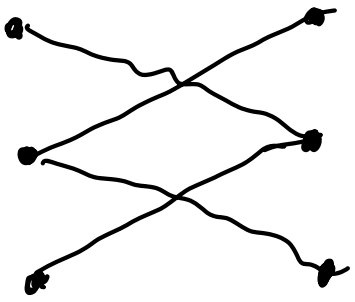
$$\Pr_{h \sim \{0,1\}^{poly(n)}}[A(x,h) = True] \begin{cases} > 0 & x \in L \\ = 0 & x \notin L \end{cases}$$

Example: VertexCover

$\langle G=(V,E), k \in \mathbb{N} \rangle \in L$

iff there's a vertex cover $S \subseteq V$

s.t. $|S| \leq k$, and every $e \in E$ has endpoint $\in S$



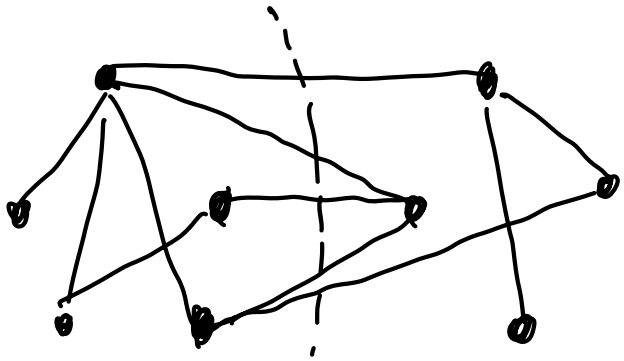
, 2 ✓

Example: Max Cut

$\langle G=(V,E), k \in \mathbb{N} \rangle \in L$

iff there's a cut $S \subseteq V$, $V \setminus S$

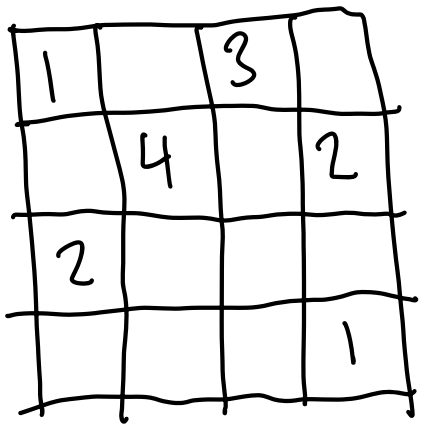
s.t. (# edges crossing between $S, V \setminus S$) $\geq k$



, 5 ✓

Example: Sudoku

$\langle$ game state $\rangle \in L$ iff possible to extend current game state to full board w/ 1-n in each row/col/subgrid



NP has access to powerful tools that P doesn't have.

Million dollar question: $P = NP$?

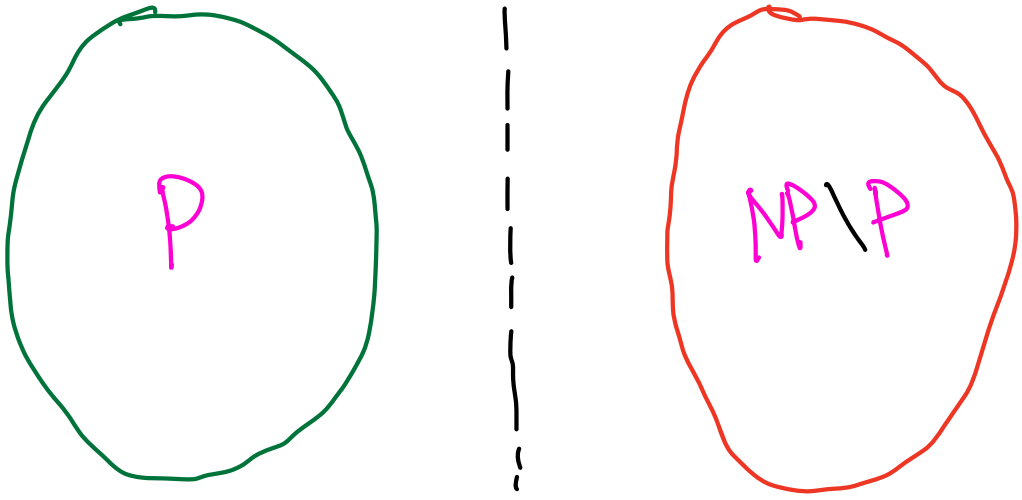
NP-hardness (Part VIII, Section 3.2)

We say a language L^* is NP-hard if

$$\forall L \in NP,\ L \leq_P L^*$$

"given access to  can simulate $P = NP$"

We say L is NP-complete $\iff$ $L \in NP$ and $L \in$ NP-hard



NP-complete
= bottleneck to $P = NP$

View of reductions & hardness:

If $A \leq_P B$, we say B is "harder" than A

There is a world where $A \in P$, $B \notin P$

There is no world where $B \in P$, $A \notin P$

NP-complete: hardest problems in NP!

*from perspective of proving $P = NP$

We don't know if we live in

Algorithmica: $P = NP$

Pessiland*: $P \neq NP$

* Simplification of language

But we can prove certain problems, e.g. SAT, are in NP & NP-hard! Why useful?

NP-complete

In 2019, 88-99% believe we live in Pessiland. Assuming this is true,

ANY NP-complete L can't be in P!

The theory is useful because we can prove many interesting problems are NP-complete.

(next lecture + HW)

e.g.

- SAT
- 3SAT
- Independent Set
- Vertex Cover
- Subset Sum
- Subgraph Iso
- Partition
- Knapsack

} lecture / HW

- Clique
- Coloring
- Ham Cycle

} Erickson

- Longest Path
- Max Cut
- TSP
- Sudoku
- Cookie Clicker

} ...

What if we live in Algorithmica?

Also useful:

Better SAT $\Rightarrow$ better algos for all of NP !!!

Bad news: much of cryptography based on assumption that we don't live in Algorithmica.

Cook-Levin Theorem (Part VIII, Section 3.3)

Statement: SAT $\in$ NP-hard

What is SAT? (Satisfiability)

Input: $\langle$ Boolean formula Φ $\rangle$

Output: True or False is Φ satisfiable?

Φ assumed to be in CNF (Conjuctive normal form)

Literals: $x_1, x_2, \dots, x_n$ or negations

Φ = conjunction of clauses (AND of ORs)

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

$\vee$ = OR

$\wedge$ = AND

$\neg$ = NOT

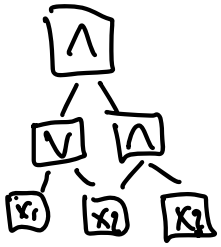
Set literals $\in \{$True, False$\}$ to make Φ = True?

Rough intuition of proof:

1) SAT $\approx$ Circuit-SAT

Boolean formula

Boolean circuit



gates = $\wedge, \vee, \neg$

2) Given $L \in NP$, $x \in \{0,1\}^n$

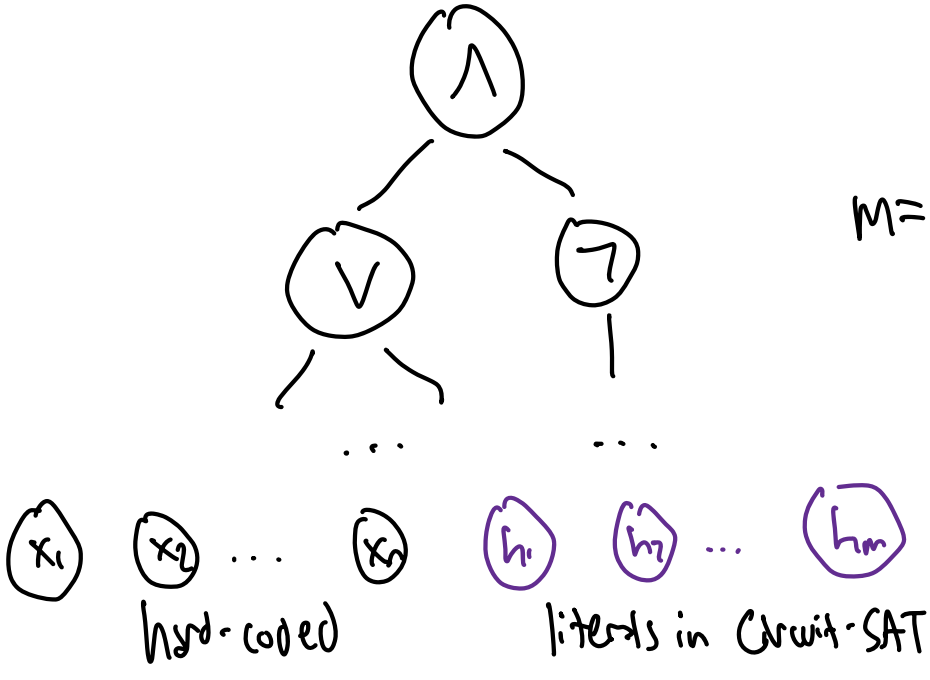
build circuit C_x s.t. $x \in L$

$\iff$ C_x is satisfiable

Idea: $\exists A$ s.t. $A(x, h) =$ True for some h

iff $x \in L$

satisfiable?



$m = \text{poly}(n)$

hard-coded

literals in Circuit-SAT