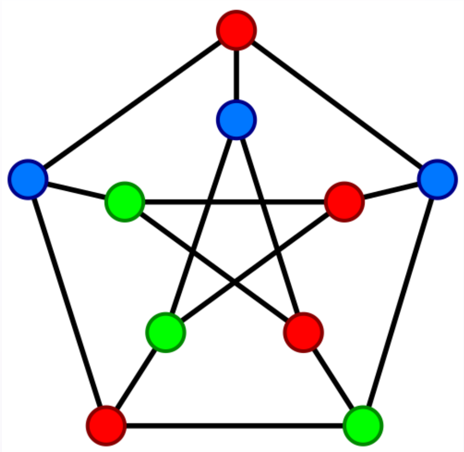
CSE 417 Autumn 2025

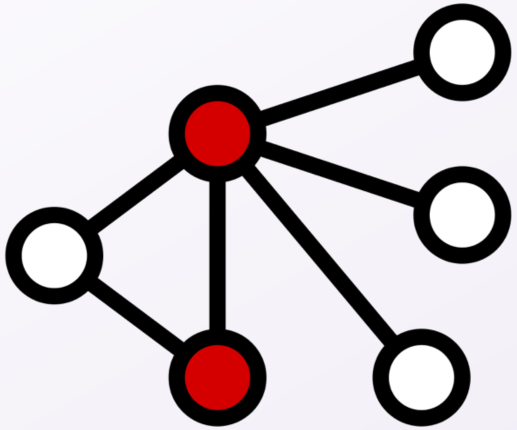
Lecture 24: Intro to NP-completeness

Glenn Sun

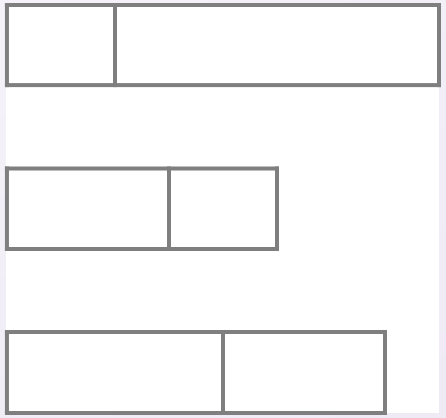
Some problems are hard to solve exactly



graph coloring



vertex cover



load balancing

What to do when a problem is hard?

Idea 1: Try a brute-force algorithm

- Okay if you know the inputs are going to be very small

Idea 2: Try an approximation algorithm or a greedy idea

- May still be quite far from optimal

Idea 3: Try to use a SAT solver

What do easy and hard mean?

decision problem: answer is “yes” or “no”

P: decision problems that can be solved in $O(n^c)$ time for some c (here, n is the input size)

easy ≈ P

hard ≈ everything not in P

A bit of practice

Do the following problems belong to P?

Q: Find the smallest number of coins needed to make change with quarters, dimes, nickels, and pennies.

A: No, not a decision problem!

A bit of practice

Do the following problems belong to P?

Q: Is there a spanning tree with weight $\leq \boldsymbol{k}$?

A: Yes, run Prim's or Kruskal's algorithms, then compare with$\boldsymbol{k}$.

A bit of practice

Do the following problems belong to P?

Q: Given a graph, can it be colored with k colors so that no edge has the same color on both endpoints?

A: We don't know! (We don't think so.)

The class NP

NP: decision problems whose solutions can be *checked* in $O(n^c)$ time for some c

Example: Given a sample coloring $\mathbf{color}(v)$, can check if graph is correctly colored with k colors in linear time.

- Loop through all v, check that $color(v)$ uses at most k colors
- Loop through all edges (u, v), check that $\mathbf{color}(u) \neq \mathbf{color}(v)$

P vs. NP

Most believe this

Most important open problem in CS:

Prove that either P = NP or P ≠ NP.

In other words, either:

- Show how to solve every problem quickly, only knowing that it can be checked quickly, or
- Give an example of a problem that can be checked quickly, but cannot be solved quickly.

How to prove not in P?

There are some problems that cannot be solved in polynomial time.

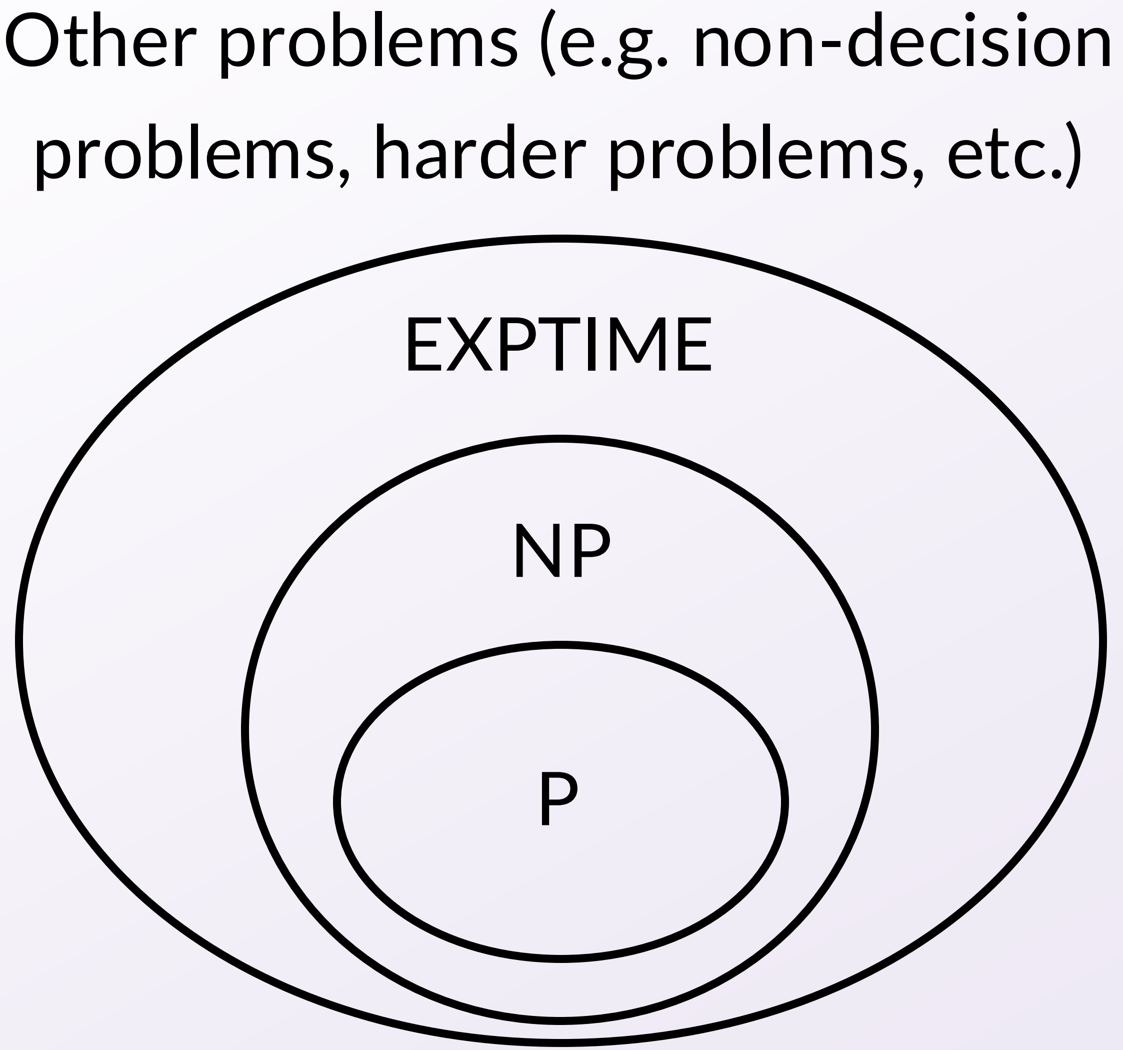
- Given a piece of code, an input to the code, and a number n, determine if the code will terminate within n "steps".

Exponential time algorithm: Run the code for n steps. But n is a number, taking $\log(n)$ bits in the input, so this is exponential.

Why impossible in P? Wait for Wednesday, December 3!

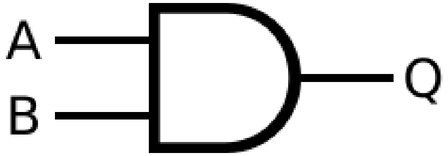
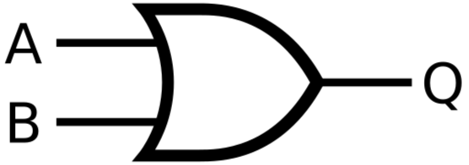
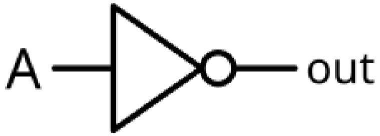
Does **not** resolve P vs. NP because seems hard to verify answers.

What people think the world looks like



Working with Boolean formulas

Different notations in different fields

<table>
<tr><th>Operation</th><th>Math/CS Theory</th><th>Programming</th><th colspan="2">Electrical Engineering</th></tr>
<tr><td>and</td><td>$a \wedge b$</td><td>a && b</td><td>AB</td><td>
</td></tr>
<tr><td>or</td><td>$a \vee b$</td><td>a || b</td><td>$\mathrm{A} + \mathrm{B}$</td><td>
</td></tr>
<tr><td>not</td><td>$\neg a$</td><td>!a or ~a</td><td>$\overline{\mathrm{A}}$</td><td>
</td></tr>
</table>

Truth tables

and

a	b	$a \land b$
0	0	0
0	1	0
1	0	0
1	1	1

or

a	b	$a \lor b$
0	0	0
0	1	1
1	0	1
1	1	1

not

a	$\neg a$
0	1
1	0

Truth tables

implies

a	b	$a \Rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

if and only if

a	b	$a \Leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

exclusive or (xor)

a	b	$a \bigoplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Practice with truth tables

"Do $a \Rightarrow b$ and $\neg a \vee b$ mean the same thing?"

a	b	$a \Rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

a	b	$\neg a$	$\neg a \vee b$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

Yes!

Rewriting expressions

Q: Rewrite $\neg(a \wedge b)$ to an expression where $\neg$ only appears on the "inside" (attached to variables, not larger expressions).

A: $\neg a \vee \neg b$

a	b	$a \wedge b$	$\neg(a \wedge b)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

a	b	$\neg a$	$\neg b$	$\neg a \vee \neg b$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

De Morgan’s laws

$$\neg(a \vee b) = \neg a \wedge \neg b$$

$$\neg(a \wedge b) = \neg a \vee \neg b$$

Distributivity laws

Just like $a \times (b + c) = (a \times b) + (a \times c)$, similar things are true for boolean expressions:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

All about SAT

2SAT

Recall 2SAT from the graph algorithms unit:

Input: A set of implications of the form "if $\boldsymbol{a}$, then $\boldsymbol{b}$"

Goal: Determine if all implications can be simultaneously satisfied

Since $\boldsymbol{a} \Rightarrow \boldsymbol{b}$ and $\neg\boldsymbol{a} \vee \boldsymbol{b}$ are equivalent, this is the same as:

Input: A set of *clauses* $\boldsymbol{a} \vee \boldsymbol{b}$ where $\boldsymbol{a}$ and $\boldsymbol{b}$ are possibly negated

Goal: Determine if all clauses can be simultaneously satisfied

SAT (Satisfiability)

literals: variables or their negation

$$a, \neg b, x, \neg x, y$$

clause: OR of literals

$$(a \lor \neg b), (x \lor \neg y \lor z)$$

conjunction normal form (CNF): AND of clauses

$$(a \lor \neg b) \land (x \lor \neg y \lor z)$$

SAT (Satisfiability)

Input: A CNF formula $f(x_1, \ldots, x_n)$ (equivalently a set of clauses)

Goal: Does there exist $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n)$ is true?

2SAT: Clauses in the input are restricted to length 2.

k**-SAT:** Clauses in the input are restricted to length k.

SAT example

Q: Is the following CNF satisfiable? Why or why not?

$$
\begin{aligned}
&(\neg a \lor b \lor d) \land (\neg b \lor c \lor d) \land (a \lor \neg c \lor d) \land (a \lor \neg b \lor \neg d) \\
&\qquad \land (b \lor \neg c \lor \neg d) \land (\neg a \lor c \lor \neg d) \land (a \lor b \lor c) \land (\neg a \lor \neg b \lor \neg c)
\end{aligned}
$$

A: No, not satisfiable.

The importance of SAT

SAT was the first problem to be shown to be **NP-hard**.

This means that: *every* NP problem can be encoded as an instance of SAT with a small (polynomial overhead).

In other words, if you could find an algorithm to solve SAT, you automatically have an algorithm to solve all NP problems!

Similar to: Using graph algorithms to solve new problems, using Ford–Fulkerson to model new problems, etc.

The importance of SAT

- General proof for all NP problems: Cook–Levin theorem, 1971

We won't cover this, requires a mathematical definition of "what is an algorithm" and "what is a computer" (Turing machines).

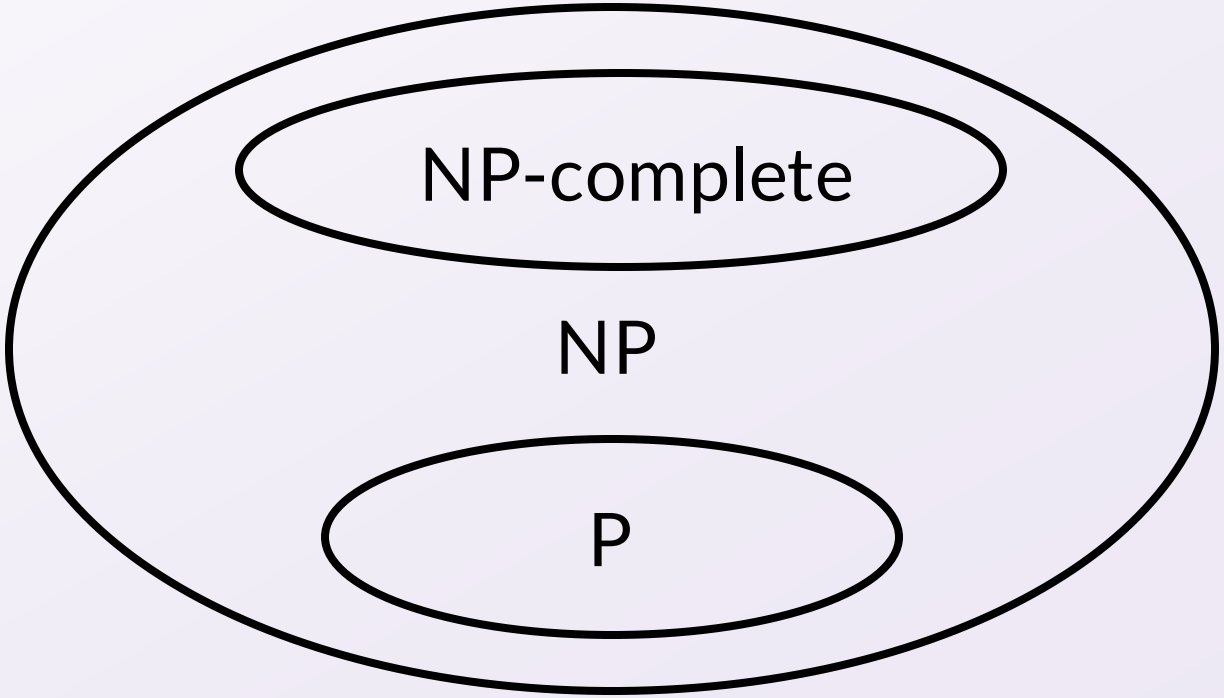
- Specific cases for some particular NP problems: next class!

NP-completeness

A problem that is both in NP and NP-hard is called **NP-complete**, such as SAT.

These are the hardest problems in NP.

What we believe
(assuming P ≠ NP)



Final reminders

HW5 (DP) resubmissions close tonight @ 11:59pm!

HW7 (Flows) due **next** Wednesday night

I have OH now-12:30pm:

- Meet at front of classroom, we’ll walk over together
- CSE (Allen) 214 if you’re coming later

Nathan has online OH 12–1pm:

- https://washington.zoom.us/my/nathanbrunelle