CS314 - NP-Hardness
“Efficiency"

Fundamental question: Are there hard problems? How hard—unsolvable? polynomial?
Undecidability.

Some problems can not be solved:

Halting Problem: Given a program $P$ and input $I$, does $P$ halt given input $I$ or does it run forever?

Output: true or false

Why useful?
Note: Our program can't just simulate and running on I.

Why? If we simulate P on I and it runs forever, we don't actually output.
Thm. [Turing '36]: The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: (by contradiction)
Suppose we have such a program \( h \):
\[
    h(P, I) =
    \begin{cases} 
        1 & \text{if } P \text{ halts on } I \\
        0 & \text{otherwise}
    \end{cases}
\]
Now define another program:

\[ g(i) : \]
\[
\text{if } h(i, i) = 0 \\
\text{return } 0 \\
\text{else loop forever}
\]

try \( g(g) \rightarrow h(g, g) \)
Now what does g(g) do?

calls h(g, g):

\[ h(g, g) \rightarrow \text{if } 1, \text{ then } g \text{ halts on input } g. \]

Then g(g) should loop forever.

\[ h(g, g) \text{ outputs } 0 \text{, then } \text{it should halt.} \]

This means it looped forever, \( g \) cannot exist.
So... what next?

Clearly, lots of things are doable in polynomial time.

Some things are impossible.

But - is there anything in between?

Idea: exponential time

no $n^c$ (poly)

$2^n \cdot n$
Candidate: Circuits

Boolean gates

An AND gate, an OR gate, and a NOT gate.

- No loops
- Given inputs, can calculate output in linear time; basically evaluate in BFS order
Q: Given a boolean circuit, is there a set of inputs that evaluate to true?

Circuit satisfiability (Circuit SAT)

true
Best known algorithm:
try all $2^m$ possible inputs.

Running time: $2^m(n+m)$

The current best-known bound is no proof stating it couldn't be done faster.
$P \subseteq NP \subseteq co-NP$

Consider decision problems: Yes or No.

$P$: set of decision problems that can be solved in polynomial time.

Ex.: is this list sorted?

NP: set of problems st. if the answer is yes, this can be checked in poly. time.

(So can verify a yes answer.)
Ex: Circuit SAT
given set of boolean inputs
can check that they give
yes answer.

co-NP: set of problems where
we can check a no answer
in poly time.
NP-Hard

$\Pi$ is NP-hard $\iff$ If $\Pi$ can be solved in polynomial time, then $P=NP$

So if an NP-Hard problem can be solved in polynomial time, then any problem in NP can be solved in polynomial time.

(Paths story ...)

$P$ vs $NP$
A problem is NP-Complete if it is both:
- in NP
- NP-Hard

More of what we *think* the world looks like.

polynomial hierarchy
To prove NP-Hardness of A:

Reduce a known NP-Hard problem to A.
bipartite matching

$G \rightarrow \text{changed to a flow network } G'$

$G' \rightarrow \text{max flow alg (of size } k\text{)}$
So to prove your problem is hard, solve a different problem using your problem as a subroutine.

Cook's Theorem: Circuit SAT is NP-Hard.

(Just trust me)
**Def:** SAT takes a boolean formula $\phi$ and asks if it is possible to assign booleans so the formula is true.

**Ex:** \((a \lor b \lor c \lor \overline{d}) \leftrightarrow ((b \land \overline{c}) \lor (\overline{a} \Rightarrow d) \lor (c \neq a \land b))\)

$m$ variables, $n$ clauses

In $NP$: given assignment $a, b, c, d$, can check if it evaluates to true in $O(m+n)$ time.
Thm: SAT is NP-hard.

Pf: Reduction:

Reduce circuit SAT to SAT
So our reduction looks like this:

boolean circuit $\xrightarrow{O(n)}$ boolean formula

trivial $\xrightarrow{\text{SAT}}$ TRUE or FALSE

or: