NP = Non-deterministic Polynomial Time
is a large family of problems
expect some are <u>not</u> solvable in polynomial time.

NP-hard: Problems at least as hard as <u>everything</u>
in NP.
If can solve one in polynomial time,
can solve everything in NP in polynomial time

NP-complete: NP-hard and in NP
essentially equivalent to each other
The hardest problems in NP.

Idea behind NP: (motivation before definition)
Problems for which it might be hard to find the
answer. But once found is easy to check.

Like systems of equations:
takes work to find solution, but easy to check.

Linear equations are in polynomial time,
but Systems of Polynomial Equations are hard.
Abbreviate SPE.

SPE: Have variables, say $x_1, \dots, x_n$,
and polynomials $p_1(x), \dots, p_k(x)$.

Problem 1: find x s.t. $p_i(x) = 0$ for all i?

Problem 2: Does there exist x s.t. $p_i(x) = 0$ for all i?

Problem 1: Given this x, can efficiently check if satisfies all equations. If x is rational, and the coefficients of the polynomials are rational, can check.

issue i: $x^2 - 2 = 0$ does not have a rational solution

issue ii: What if the solution is much larger than the problem? That is, if $\text{size}(x) \gg \text{size}(p_1, \dots, p_k)$? From the perspective of the problem, this is inefficient.

issue iii: What if there is no solution? Is that an answer? How could we check that?

We go with Problem 2, which has just yes/no answers.
If "yes", is an x that you can (try to) check.
If "no", there might not be.

Problems with yes/no answers are called decision problems

An NP-complete problem: $\{0,1\}$-SPE

Does there exist $x \in \{0,1\}^n$ s.t. $p_i(x) = 0$, $\forall i$?

Now, the solution can <u>not</u> be big: is in $\{0,1\}^n$.

Can evaluate $p_i(x)$ in time polynomial in $\text{size}(p_i)$, so can check x efficiently.

For yes answers and x proving answer is "yes" $p_i(x) = 0$ $1 \le i \le k$, call x a witness, certificate, or proof.

For no answers there does not need to be.

Let $q = (p_1 \dots p_k)$ specify the problem.

Write $q \in \{0,1\}$-SPE if is valid problem with yes answer.

<u>Def</u> A problem Y (like $\{0,1\}$-SPE) is in NP if $\exists$ a polynomial-time algorithm $\mathcal{A}$ (witness checker) and constant c governing answer size such that for all q (problem instances)

if $q \in Y$ (valid and yes answer)

$\exists\, \omega$ (witness) s.t. $\mathcal{A}(q, \omega) =$ "yes", and $\text{size}(\omega) \le c\, \text{size}(q)^c$

if $q \notin Y$, $\forall\, \omega$ s.t. $\text{size}(\omega) \le c\,\text{size}(q)^c$ $\mathcal{A}(q, \omega) =$ "no"

Key points:

If answer is yes, $\exists w$ that convinces $\mathcal{A}$

" " " no, $\mathcal{A}$ will not say "yes"

If q is not valid problem instance, $\mathcal{A}$ says "no"

$\mathcal{A}$ runs in time polynomial in size(q) and size(w).

w could be larger than q (rarely is)

But size(w) is polynomial in size(q)

So time of $\mathcal{A}$ is polynomial in size(q).

$\{0,1\}$-SPE is in NP

Def. A problem Υ is in P if

$\exists$ a polynomial-time algorithm $\mathcal{A}$ s.t.

$q \in \Upsilon \Rightarrow \mathcal{A}(q) =$ "yes"

$q \notin \Upsilon \Rightarrow \mathcal{A}(q) =$ "no"

Linear feasibility: $\exists x$ s.t. $Ax \leq b$? is in P

Can turn optimization problems into decision problems:

ask if $\exists x$ s.t. $f(x) \leq t$ and $g(x) \leq 0$.

Then do search on t.

Can learn x by asking about its bits.

All this is in NP.

"Algorithm" is a bit vague. Program is more precise.
Turing machines formalize this.
We will use logic circuits.

NP contains very hard problems

- Factoring
- Breaking any public-key cryptography
- Design anything given specifications.
- Prove any theorem whose proof is not too long

How do we prove something is NP-hard?

Reductions.

A Karp reduction from Y to Z is a polynomial time algorithm $\mathcal{A}$ s.t.

$$q \in Y \quad \text{iff} \quad \mathcal{A}(q) \in Z$$

$\mathcal{A}$ transforms problem Y into problem Z

Given $\mathcal{A}$ and a Ptime algorithm for Z,
can solve Y in Ptime.

If Y is hard, then Z must also be hard.

A Cook reduction from Y to Z is a polynomial time algorithm $\mathcal{A}$ that decides if $q \in Y$
using an Oracle for Z that decides membership in Z
in constant time.

Can solve Y given a subroutine for Z

Y is polynomial-time reducible to Z "$Y \leq_P Z$"
if $\exists$ Cook reduction from Y to Z
Karp reductions are Cook reductions,
and always use Karp reductions
(For all known decision problems in NP
with $Y \leq_P Z$, is a Karp reduction)

Z is NP-hard if $\forall Y \in NP$, $Y \leq_P Z$.

Still seems like a lot to show.
NP-complete problems make this manageable.

Z is NP-complete (NPC)
if $Z \in NP$ and Z is NP-hard.

To show X is NP-hard, just show $Z \le_p X$ for some NP-hard Z.

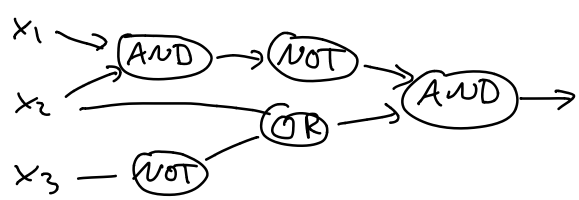
Then $\forall Y \in NP$, $Y \le_p Z$ and $Z \le_p X \Rightarrow Y \le_p X$
$\Rightarrow X$ is NP-hard

Once we know one such Z, always work from it and never have to write $\forall Y \in NP$ again.

<u>Theorem</u> Circuit-Satisfiability (C-SAT) is NP-complete.

Problem: given a logic circuit with one output, is there an input that makes the output true?

Example



A binary boolean circuit has gates numbered $1, \dots, k$ s.t. each is either

a. an input (True/False 1/0)

b. the negation (NOT) of a lower numbered gate

c. the AND or OR of two lower numbered gates.

Gate k is the output.

Wolog inputs are gates 1 through n, and $k \ge n$.

Observation: can view every gate g_j as a function of the inputs. If all these equations are satisfied, $y_j = g_j(x_1, \ldots x_n)$, for all j.

Why C-SAT is NP-Complete:

C-SAT $\in$ NP : given an input can evaluate every gate.

C-SAT is NP-hard because (roughly)

for every algorithm A that runs in time $T(n)$ on inputs of size n,

For every n there is a circuit C_n with $\leq O((T(n)+n)^2)$ gates s.t.

$C_n(x) = A(x)$ for all $x \in \{0,1\}^n$

Anything a ptime algorithm can do a polynomial size circuit can, too.

Now, want to prove $\{0,1\}$-SPE is NP-complete and SPE is NP-hard.

We already argued $\{0,1\}$-SPE $\in$ NP.

Thml Now need to prove C-SAT $\leq_p$ $\{0,1\}$-SPE

Let q be an input to C-SAT. That is a circuit.
We need to translate into an instance of $\{0,1\}$-SPE

Let $g_1 \dots g_k$ be the gates in q,
with $g_1, \dots, g_n$ being the inputs, $x_1, \dots, x_n$

Our instance of $\{0,1\}$-SPE will have variables $y_1, \dots, y_k$, and $k-n+1$ equations.

If $g_j = \text{NOT}(g_i)$ we add equation $y_j = 1 - y_i$

$g_j = \text{AND}(g_h, g_i)$ " " " $y_j = y_h y_i$

for $y_h, y_i \in \{0,1\}$ this is satisfied iff $y_j = \text{AND}(y_h, y_i)$

$g_j = \text{OR}(g_h, g_i)$, add equation $y_j = y_h + y_i - y_h y_i$

To ensure g_k outputs true, add equation $y_k = 1$

To put these in proper form, rewrite as

NOT: $y_j + y_i - 1 = 0$

AND $y_j - y_h y_i = 0$

OR: $y_j - y_h - y_i + y_h y_i = 0$

Output $y_k - 1 = 0$

So, if $\exists\, x_1, \dots, x_n$ s.t. $g_k(x_1 \dots x_n) = 1$,
setting $y_j = g_j(x_1 \dots x_n)$, and $y_i = x_i \quad 1 \le i \le n$

results in $y_1 \dots y_k$ satisfying all equations.

Conversely, if $y_1 \dots y_k$ satisfy all equations
$g_k(y_1 \dots y_n) = 1$

We have reduced any circuit into a very special system of quadratic equations. This sort of system is hard to solve.

Thm 2 $\quad \{0,1\}\text{-SPE} \le_p \text{SPE}$
so SPE is NP-hard
proof use same equations on $x_1 \dots x_n$, but add $x_i(1-x_i) = 0$, $\forall i$
Is satisfied iff $\forall\, x_i \in \{0,1\}$

SPE is hard, and probably not in NP.
(in fact, we know is not).