<table>
<thead>
<tr>
<th>Hard problems (NP-complete)</th>
<th>Easy problems (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>2SAT, Horn SAT</td>
</tr>
<tr>
<td>TSP</td>
<td>Min. spanning tree</td>
</tr>
<tr>
<td>Longest path</td>
<td>Shortest path</td>
</tr>
<tr>
<td>3D matching</td>
<td>Bipartite matching</td>
</tr>
<tr>
<td>Knapsack</td>
<td>Unary knapsack</td>
</tr>
<tr>
<td>Ind. set</td>
<td>Ind. set on trees</td>
</tr>
<tr>
<td>ILP</td>
<td>LP</td>
</tr>
<tr>
<td>Hamiltonian cycles</td>
<td>Euler paths</td>
</tr>
<tr>
<td>Balanced cut</td>
<td>Minimum cut</td>
</tr>
</tbody>
</table>
POLY-TIME VERIFICATION

• specified by an algorithm $C$
• inputs:
  ▪ instance $I$
  ▪ proposed solution $S$
• runs in time polynomial in $|I|$
• $S$ is a solution to $I$ iff $C(I, S) = true$
WHAT ARE P AND NP?

- P: problems solvable in polynomial time
- NP: problems verifiable in polynomial time
WHAT ARE P AND NP?

- P: "polynomial time"
- NP: "nondeterministic polynomial time"
• P: problems solvable in polynomial time
• NP: problems verifiable in polynomial time

So $P \subseteq NP$
Are all poly-time verifiable problems solvable in polynomial time?

Does $P = NP$?

Or is $P \subsetneq NP$?
MILLENIUM PROBLEMS

- Yang–Mills and Mass Gap
- Riemann Hypothesis
- P vs NP Problem
- Navier–Stokes Equation
- Hodge Conjecture
- Poincaré Conjecture
- Birch and Swinnerton-Dyer Conjecture
If we reduce $A$ to $B$, we write $A \rightarrow B$.

If we could efficiently solve $B$, we could solve $A$.

If we know $A$ cannot be efficiently solved, then neither can $B$!

($B$ is at least as hard as $A$)
If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

Given $(f_{AB}, h_{AB})$ and $(f_{BC}, h_{BC})$, we can compose them:

$$(f_{BC} \circ f_{AB}, h_{AB} \circ h_{BC})$$
NP-COMPLETENESS

A problem is NP-complete if:

- it's in NP
- all other problems in NP reduce to it

(NP-complete problems are the "hardest" in NP)
To show $B$ is NP-complete:

- Given an NP-complete problem $A$,
- Reduce $A$ to $B$
Increasing difficulty
ASIDE: FACTORING

Can we factor a $b$-bit number in time $O(b^k)$?

- Can verify solution easily (in NP)
- Seems to be hard to solve (not in P?)
- Not known to be NP-complete
HAM. PATH $\rightarrow$ HAM. CYCLE

Is it easier to find a cycle than an $(s, t)$ path?
HAM. PATH $\rightarrow$ HAM. CYCLE
**RUDRATA \((s, t)\)-PATH**

**Instance:**

\[ G = (V, E) \]

nodes \(s, t\)

Add node \(x\) and edges \(\{s, x\}, \{x, t\}\)

\[ G' = (V', E') \]

**RUDRATA CYCLE**

Solution: cycle \(\{s, x\}, \{x, t\}\)

Delete edges \(\{s, x\}, \{x, t\}\)

No solution

Solution: path

No solution
• If a cycle is found
  ▪ Delete edges \((t, x)\) and \((x, s)\)
• If not
  ▪ Then there's no path
  ▪ (Contrapositive: \((s, t)\)-path \(\Rightarrow\) cycle)
3SAT $\rightarrow$ IND. SET

$$(\overline{x} \lor y \lor z)(x \lor \overline{y} \lor z)(x \lor y \lor z)(\overline{x} \lor \overline{y})$$
• If an independent set is found
  ■ Make those values true
• If not
  ■ Then there's no satisfying assignment
  ■ (Contrapositive: assignment $\Rightarrow$ independent set)
SAT $\rightarrow$ 3SAT

$$(\overline{x} \lor y \lor z)(x \lor y \lor \overline{z} \lor \overline{w} \lor u \lor v)(w \lor z)(\overline{y} \lor \overline{w})$$
\[
(a_1 \lor a_2 \lor y_1) (\overline{y}_1 \lor a_3 \lor y_2) (\overline{y}_2 \lor a_4 \lor y_3) \cdots (\overline{y}_{k-3} \lor a_{k-1} \lor a_k)
\]
\[
\{ (a_1 \lor a_2 \lor \cdots \lor a_k) \text{ is satisfied} \} \iff \{ (a_1 \lor a_2 \lor y_1) (\overline{y}_1 \lor a_3 \lor y_2) \cdots (\overline{y}_{k-3} \lor a_{k-1} \lor a_k) \text{ are all satisfied} \}
\]
\[
\begin{align*}
\{ (a_1 \lor a_2 \lor \cdots \lor a_k) \text{ is satisfied} \} & \iff \left\{ \begin{array}{l l}
\text{there is a setting of the } y_i \text{'s for which } \\
(a_1 \lor a_2 \lor y_1) (\overline{y}_1 \lor a_3 \lor y_2) \cdots (\overline{y}_{k-3} \lor a_{k-1} \lor a_k) \text{ are all satisfied} \\
\end{array} \right.
\end{align*}
\]

- If the RHS is satisfied
  - At least one of the $a_i$ values must be true
- If the LHS is satisfied
  - Then some $a_i$ is true
  - Set $y_1, \ldots, y_{i-2}$ to true
  - Set the other $y_j$ to false
ANY NP PROBLEM $\rightarrow$ SAT

- Any NP problem $\rightarrow$ Circuit SAT
- Circuit SAT $\leftrightarrow$ SAT
CIRCUIT SAT

output

AND

NOT

OR

AND

true

?
?

?
SAT $\rightarrow$ CIRCUIT SAT

- AND gates at the top
- OR gates in each clause
- literals are (NOT of) unknowns
CIRCUIT SAT $\rightarrow$ SAT
\( g \)  
\[
\text{OR}  
\]
\[
h_1 \quad h_2
\]
\[
(g \lor \overline{h}_2)  
(g \lor \overline{h}_1)  
(\overline{g} \lor h_1 \lor h_2)
\]

\( g \)  
\[
\text{AND}  
\]
\[
h_1 \quad h_2
\]
\[
(\overline{g} \lor h_1)  
(\overline{g} \lor h_2)  
(g \lor \overline{h}_1 \lor \overline{h}_2)
\]

\( g \)  
\[
\text{NOT}  
\]
\[
h
\]
\[
(g \lor h)  
(\overline{g} \lor \overline{h})
\]
ANY NP PROBLEM $\rightarrow$ CIRCUIT SAT

- Given problem $A$ in NP
- $C(I, S)$ verifies solution $S$ in polynomial time
- Can be converted to a circuit with polynomial number of gates
- Bits of $S$ become unknowns
- Then satisfying assignments to unknowns $\iff$ solutions of $I$
NP-COMPLETENESS

A problem is NP-complete if:

- it's in NP
- all other problems in NP reduce to it

(NP-complete problems are the "hardest" in NP)
NP-COMPLETENESS

If we could solve even one NP-complete problem in polynomial time,

- We could solve any other problem in NP!
- Then $P = NP$
ASIDE

Given a program $P$ and input $x$, can we write an algorithm to see if $P$ will halt on input $x$?

terminates(p, x)
ASIDE

Let's define another function:

```python
paradox(z):
    if terminates(z, z):
        infinite_loop()
```
paradox(z):
    if terminates(z, z):
        infinite_loop()

What if we then run this?

paradox(paradox)
What if we then run this?

paradox(paradox)

- If terminate says paradox halts, it runs forever
- If terminate says paradox runs forever, it halts
UNSOLVABLE PROBLEMS

- The halting problem is undecidable
- If $\text{Halting } \rightarrow A$, 
  - if we had an algorithm for $A$, 
  - we could use that to solve the halting problem 
  - so $A$ must also be undecidable!