NP and NP-Complete

CMSC 451, Summer 2009

8.3 Definition of NP
Decision Problems

Decision problem.
- $X$ is a set of strings.
- Instance: string $s$.
- Algorithm $A$ solves problem $X$: $A(s) = \text{yes}$ iff $s \in X$.

Polynomial time. Algorithm $A$ runs in poly-time if for every string $s$, $A(s)$ terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial.

PRIMES: $X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots \}$

Definition of $P$

$P$. Decision problems for which there is a poly-time algorithm.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE</td>
<td>Is $x$ a multiple of $y$?</td>
<td>Grade school division</td>
<td>51, 17</td>
<td>51, 16</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are $x$ and $y$ relatively prime?</td>
<td>Euclid (300 BCE)</td>
<td>34, 39</td>
<td>34, 51</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is $x$ prime?</td>
<td>AKS (2002)</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td>EDITDISTANCE</td>
<td>Is the edit distance between $x$ and $y$ less than 5?</td>
<td>Dynamic programming</td>
<td>neither</td>
<td>acgggt ttttta</td>
</tr>
</tbody>
</table>
| LSOLVE     | Is there a vector $x$ that satisfies $Ax = b$? | Gauss-Edmonds elimination | \[
\begin{bmatrix}
0 & 1 & 1 \\
2 & 4 & -2 \\
0 & 3 & 13
\end{bmatrix}
\]
|             |                              | \[
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 1 \\
0 & 1 & 1
\end{bmatrix}
\] |
Certification algorithm intuition.
- Certifier views things from "managerial" viewpoint.
- Certifier doesn’t determine whether \( s \in X \) on its own; rather, it checks a proposed proof \( t \) that \( s \in X \).

Def. Algorithm \( C(s, t) \) is a certifier for problem \( X \) if for every string \( s \), \( s \in X \) iff there exists a string \( t \) such that \( C(s, t) = \text{yes} \).

NP. Decision problems for which there exists a poly-time certifier.

Remark. NP stands for nondeterministic polynomial-time.
Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer $s$, is $s$ composite?

**Certificate.** A nontrivial factor $t$ of $s$. Note that such a certificate exists iff $s$ is composite. Moreover $|t| \leq |s|$.

**Certifier.**

```java
boolean C(s, t) {
    if (t ≤ 1 or t ≥ s)
        return false
    else if (s is a multiple of t)
        return true
    else
        return false
}
```

**Instance.** $s = 437,669$.

**Certificate.** $t = 541$ or $809$. — $437,669 = 541 \times 809$

**Conclusion.** COMPOSITES is in NP.

Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula $\Phi$, is there a satisfying assignment?

**Certificate.** An assignment of truth values to the $n$ boolean variables.

**Certifier.** Check that each clause in $\Phi$ has at least one true literal.

**Ex.**

\[
(\overline{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_1) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x}_1 \lor \overline{x}_3 \lor \overline{x}_4)
\]

**Instance $s$**

\[
x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1
\]

**Certificate $t$**

**Conclusion.** SAT is in NP.
Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** Given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that visits every node?

**Certificate.** A permutation of the $n$ nodes.

**Certifier.** Check that the permutation contains each node in $V$ exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** HAM-CYCLE is in NP.

---

**P, NP, EXP**

**P.** Decision problems for which there is a poly-time algorithm.

**EXP.** Decision problems for which there is an exponential-time algorithm.

**NP.** Decision problems for which there is a poly-time certifier.

**Claim.** $P \subseteq NP$.

**Pf.** Consider any problem $X$ in $P$.
- By definition, there exists a poly-time algorithm $A(s)$ that solves $X$.
- Certificate: $t = \varepsilon$, certifier $C(s, t) = A(s)$.

**Claim.** $NP \subseteq EXP$.

**Pf.** Consider any problem $X$ in $NP$.
- By definition, there exists a poly-time certifier $C(s, t)$ for $X$.
- To solve input $s$, run $C(s, t)$ on all strings $t$ with $|t| = p(|s|)$.
- Return yes, if $C(s, t)$ returns yes for any of these.
The Main Question: P Versus NP

**Does P = NP?**  [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...
If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on P = NP?  Probably no.

The Simpson's: P = NP?

Copyright © 1990, Matt Groening
Futurama: \( P = NP? \)

Numb3rs: \( P = NP? \)
Looking for a Job?

Some writers for the Simpsons and Futurama.


8.4 NP-Completeness
Polynomial Transformation

**Def.** Problem \( X \) **polynomial reduces** (Cook) to problem \( Y \) if arbitrary instances of problem \( X \) can be solved using:
- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem \( Y \).

**Def.** Problem \( X \) **polynomial transforms** (Karp) to problem \( Y \) if given any input \( x \) to \( X \), we can construct an input \( y \) such that \( x \) is a **yes** instance of \( X \) iff \( y \) is a **yes** instance of \( Y \).

**Note.** Polynomial transformation is polynomial reduction with just one call to oracle for \( Y \), exactly at the end of the algorithm for \( X \). Almost all previous reductions were of this form.

**Open question.** Are these two concepts the same?

NP-Complete

**NP-complete.** A problem \( Y \) in \( NP \) with the property that for every problem \( X \) in \( NP \), \( X \leq_p Y \).

**Theorem.** Suppose \( Y \) is an NP-complete problem. Then \( Y \) is solvable in poly-time iff \( P = NP \).

**Pf.** \( \Rightarrow \) If \( P = NP \) then \( Y \) can be solved in poly-time since \( Y \) is in \( NP \).

**Pf.** \( \Leftarrow \) Suppose \( Y \) can be solved in poly-time.
- Let \( X \) be any problem in \( NP \). Since \( X \leq_p Y \), we can solve \( X \) in poly-time. This implies \( NP \subseteq P \).
- We already know \( P \subseteq NP \). Thus \( P = NP \).

**Fundamental question.** Do there exist "natural" NP-complete problems?
Circuit Satisfiability

**CIRCUIT-SAT.** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

The "First" NP-Complete Problem

**Theorem.** CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

**Pf.** (sketch)

- Any algorithm that takes a fixed number of bits $n$ as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

- Consider some problem $X$ in NP. It has a poly-time certifier $C(s, t)$. To determine whether $s$ is in $X$, need to know if there exists a certificate $t$ of length $p(|s|)$ such that $C(s, t) = \text{yes}$.
- View $C(s, t)$ as an algorithm on $|s| + p(|s|)$ bits (input $s$, certificate $t$) and convert it into a poly-size circuit $K$.
  - first $|s|$ bits are hard-coded with $s$
  - remaining $p(|s|)$ bits represent bits of $t$
- Circuit $K$ is satisfiable iff there exists $t$ such that $C(s, t) = \text{yes}$.
Example

Ex. Construction below creates a circuit $K$ whose inputs can be set so that $K$ outputs true iff graph $G$ has an independent set of size 2.

$G = (V, E), n = 3$

Ex. Construction below creates a circuit $K$ whose inputs can be set so that $K$ outputs true iff graph $G$ has an independent set of size 1.

$G = (V, E), n = 2$
Establishing NP-Completeness

**Remark.** Once we establish first "natural" NP-complete problem, others fall like dominoes.

**Recipe to establish NP-completeness of problem Y.**
- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that X ≤p Y.

**Justification.** If X is an NP-complete problem, and Y is a problem in NP with the property that X ≤p Y then Y is NP-complete.

**Pf.** Let W be any problem in NP. Then W ≤p X ≤p Y.
- By transitivity, W ≤p Y.
- Hence Y is NP-complete.

3-SAT is NP-Complete

**Theorem.** 3-SAT is NP-complete.

**Pf.** Suffices to show that CIRCUIT-SAT ≤p 3-SAT since 3-SAT is in NP.
- Let K be any circuit.
- Create a 3-SAT variable x<sub>i</sub> for each circuit element i.
- Make circuit compute correct values at each node:
  - x<sub>2</sub> = ¬x<sub>3</sub> ⇒ add 2 clauses: x<sub>2</sub> ∨ x<sub>3</sub>, x<sub>2</sub> ∨ ~x<sub>5</sub>
  - x<sub>1</sub> = x<sub>4</sub> ∨ ¬x<sub>5</sub> ⇒ add 3 clauses: x<sub>1</sub> ∨ x<sub>4</sub>, x<sub>1</sub> ∨ ~x<sub>5</sub>, x<sub>1</sub> ∨ x<sub>4</sub> ∨ x<sub>5</sub>
  - x<sub>0</sub> = x<sub>1</sub> ∧ x<sub>2</sub> ⇒ add 3 clauses: x<sub>0</sub> ∨ x<sub>1</sub>, x<sub>0</sub> ∨ x<sub>2</sub>, x<sub>0</sub> ∨ ~x<sub>1</sub> ∨ ~x<sub>2</sub>

- Hard-coded input values and output value.
  - x<sub>2</sub> = 0 ⇒ add 1 clause: x<sub>5</sub>
  - x<sub>0</sub> = 1 ⇒ add 1 clause: x<sub>0</sub>

- Final step: turn clauses of length < 3 into clauses of length exactly 3. ▪
Observation. All problems below are NP-complete and polynomial reduce to one another!

by definition of NP-completeness

Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.
Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]
- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

NP-completeness can guide scientific inquiry.
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.

More Hard Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Economics: computation of arbitrage in financial markets with friction.
Electrical engineering: VLSI layout.
Environmental engineering: optimal placement of contaminant sensors.
Financial engineering: find minimum risk portfolio of given return.
Game theory: find Nash equilibrium that maximizes social welfare.
Genomics: phylogeny reconstruction.
Mechanical engineering: structure of turbulence in sheared flows.
Medicine: reconstructing 3-D shape from biplane angiogram.
Operations research: optimal resource allocation.
Physics: partition function of 3-D Ising model in statistical mechanics.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency.
Statistics: optimal experimental design.