NP-COMPLETENESS

Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?


<table>
<thead>
<tr>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>Matching</td>
<td>3D-matching</td>
</tr>
<tr>
<td>Min cut</td>
<td>Max cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar 4-color</td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
</tr>
</tbody>
</table>
Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.
- Given a Turing machine, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Bad news. Huge number of fundamental problems have defied classification for decades.

Good news. These problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

Polynomial-Time Reduction

Desiderata. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces 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.

Notation. $X \leq_P Y$. 

Don't confuse with reduces from computational model supplemented by special piece of hardware that solves instances of Y in a single step.
Polynomial-Time Reduction

**Purpose.** Classify problems according to relative difficulty.

**Design algorithms.** If \( X \leq_p Y \) and \( Y \) can be solved in polynomial-time, then \( X \) can also be solved in polynomial time.

**Establish intractability.** If \( X \leq_p Y \) and \( X \) cannot be solved in polynomial-time, then \( Y \) cannot be solved in polynomial time.

**Establish equivalence.** If \( X \leq_p Y \) and \( Y \leq_p X \), we use notation \( X \equiv_p Y \).

Definition of NP

up to cost of reduction
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. \( X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots \} \)
Algorithm. [Agrawal-Kayal-Saxena, 2002] \( p(|s|) = |s|^8 \).

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>EDIT-DISTANCE</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>
<tr>
<td>LSOLVE</td>
<td>Is there a vector x that satisfies ( Ax = b )?</td>
<td>Gauss-Edmonds elimination</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 \( C(s, t) = yes \). 

"certificate" or "witness"

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

**Remark.** NP stands for nondeterministic polynomial-time.

\( C(s, t) \) is a poly-time algorithm and 
\( |t| \leq p(|s|) \) for some polynomial \( p(\cdot) \).

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 \leq 1 or t \geq 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.
Satisfiability

**Literal.** A Boolean variable or its negation.  
\[ x_i \text{ or } \overline{x_i} \]

**Clause.** A disjunction of literals.  
\[ C_j = x_1 \lor \overline{x_2} \lor x_3 \]

**Conjunctive normal form.** A propositional formula \( \Phi \) that is the conjunction of clauses.  
\[ \Phi = C_1 \land C_2 \land C_3 \land C_4 \]

**SAT.** Given CNF formula \( \Phi \), does it have a satisfying truth assignment?

**3-SAT.** SAT where each clause contains exactly 3 literals.

\[
\begin{align*}
\text{Ex.} & \quad (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (x_2 \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_3}) \\
\text{Yes.} & \quad x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}.
\end{align*}
\]

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.

\[
\text{Ex.} \quad (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor \overline{x_3}) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x_1} \lor \overline{x_3} \lor \overline{x_4})
\]

\[
\text{instance } \hat{s} \quad \begin{cases} 
\hat{x}_1 = 1, \hat{x}_2 = 1, \hat{x}_3 = 0, \hat{x}_4 = 1 
\end{cases}
\]

**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 $|t| \leq p(|s|)$.
- 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.

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

**Pf.** \( \Rightarrow \) 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?

![Circuit Diagram](image)

Yes: 1 0 1
Theorem. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Proof. (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) = 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 C(s, t) = yes.

Example: Independent Set

INDEPENDENT SET. Given a graph G = (V, E) and an integer k, is there a subset of vertices S ⊆ V such that |S| ≥ k, and for each edge at most one of its endpoints is in S?

Ex. Is there an independent set of size ≥ 6? Yes.
Ex. Is there an independent set of size ≥ 7? No.
Example: Independent Set

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.

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

$\begin{align*}
\text{u} & \quad \text{v} \\
\text{v} & \quad \text{w}
\end{align*}$

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 $X \leq_p Y$.

Justification. If $X$ is an NP-complete problem, and $Y$ is a problem in NP with the property that $X \leq_p Y$ then $Y$ is NP-complete.

Pf. Let $W$ be any problem in NP. Then $W \leq_p X \leq_p Y$.

- By transitivity, $W \leq_p Y$.
- Hence $Y$ is NP-complete. $\blacksquare$

by definition
by assumption
**3-SAT is NP-Complete**

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

**Pf.** Suffices to show that CIRCUIT-SAT \( \leq_p \) 3-SAT since 3-SAT is in NP.

- Let \( K \) be any circuit.
- Create a 3-SAT variable \( x_i \) for each circuit element \( i \).
- Make circuit compute correct values at each node:
  - \( x_2 = \overline{x}_4 \Rightarrow \) add 2 clauses: \( x_2 \lor x_3, \overline{x}_2 \lor \overline{x}_3 \)
  - \( x_4 = x_4 \lor x_5 \Rightarrow \) add 3 clauses: \( x_1 \lor \overline{x}_4, x_1 \lor \overline{x}_5, \overline{x}_1 \lor x_4 \lor x_5 \)
  - \( x_0 = x_0 \land x_2 \Rightarrow \) add 3 clauses: \( \overline{x}_0 \lor x_1, \overline{x}_0 \lor x_2, x_0 \lor \overline{x}_1 \lor \overline{x}_2 \)

- Hard-coded input values and output value.
  - \( x_5 = 0 \Rightarrow \) add 1 clause: \( \overline{x}_5 \)
  - \( x_6 = 1 \Rightarrow \) add 1 clause: \( x_0 \)

- Final step: turn clauses of length < 3 into clauses of length exactly 3. •

**NP-Completeness**

**Observation.** All problems below are NP-complete and polynomial reduce to one another!

---

**Diagram:**

- CIRCUIT-SAT \( \leq_p \) 3-SAT
- 3-SAT \( \leq_p \) INDEPENDENT SET
- 3-SAT \( \leq_p \) DIR-HAM-CYCLE
- 3-SAT \( \leq_p \) GRAPH 3-COLOR
- 3-SAT \( \leq_p \) SUBSET-SUM
- 3-SAT \( \leq_p \) VERTEX COVER
- 3-SAT \( \leq_p \) HAM-CYCLE
- 3-SAT \( \leq_p \) PLANAR 3-COLOR
- 3-SAT \( \leq_p \) SCHEDULING
- 3-SAT \( \leq_p \) SET COVER
- 3-SAT \( \leq_p \) TSP

---

11/8/2018
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 angiocardigram.
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.

Practical Applications of NP-Completeness

I can’t find an efficient algorithm, but neither can all these famous people.

Traveling Salesman Problem

**TSP.** Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

All 13,509 cities in US with a population of at least 500
Reference: http://www.tsp.gatech.edu
Traveling Salesman Problem

**TSP.** Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

Reference: http://www.tsp.gatech.edu

11,849 holes to drill in a programmed logic array
Reference: http://www.tsp.gatech.edu
Traveling Salesman Problem

**TSP.** Given a set of n cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

[Diagram of Optimal TSP tour]

Reference: [http://www.tsp.gatech.edu](http://www.tsp.gatech.edu)

---

Hamiltonian Cycle Reduces to TSP

**TSP.** Given a set of n cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

**HAMILTONIAN-CYCLE.** given a graph \( G = (V, E) \), does there exist a simple cycle that contains every node in \( V \)?

**Claim.** \( \text{HAM-CYCLE} \subseteq \text{P}, \text{TSP} \).

**Pf.**
- Given instance \( G = (V, E) \) of HAMILTONIAN-CYCLE, create n cities with distance function

\[
d(u, v) = \begin{cases} 
1 & \text{if } (u, v) \in E \\
2 & \text{if } (u, v) \notin E
\end{cases}
\]

- TSP instance has tour of length \( \leq n \) iff \( G \) is Hamiltonian.  •
### 3-Satisfiability Reduces to Independent Set

**Claim.** $3$-SAT $\leq_p$ INDEPENDENT-SET.

**Pf.** Given an instance $\Phi$ of 3-SAT, we construct an instance $(G, k)$ of INDEPENDENT-SET that has an independent set of size $k$ iff $\Phi$ is satisfiable.

**Construction.**
- $G$ contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

\[
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_4})
\]

#### 3-Satisfiability Reduces to Independent Set

**Claim.** $G$ contains independent set of size $k = |\Phi|$ iff $\Phi$ is satisfiable.

**Pf.** $\Rightarrow$ Let $S$ be independent set of size $k$.
- $S$ must contain exactly one vertex in each triangle.
- Set these literals to true. and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

**Pf.** $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size $k$.  

\[
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_4})
\]
Solving NP-Complete Problems

Satisfiability Solvers

Every problem in NP can be reduced to SAT.
So let’s design an algorithm to solve SAT!

Input: CNF formula
Output: Truth assignment that satisfies all clauses, or failure
Solution: Search
Two main approaches:
- Backtracking (e.g.: DPLL)
- Local search (e.g.: WalkSAT)

Backtracking is worst-case exponential.
Local search does not guarantee solution.
But in practice they work quite well for most instances.

Artificial Intelligence is the study of NP-complete problems.
Backtracking

Assign truth values by depth-first search
Assigning a variable deletes false literals and satisfied clauses
Empty CNF: Success
Empty clause: Failure
Additional improvements:
  • Unit propagation (unit clause forces truth value)
  • Pure literals (same truth value everywhere)

\[
(x_1 \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x_1} \lor \overline{x_2} \lor x_4)
\]

The DPLL Algorithm

```
DPLL(CNF) {
    if CNF is empty then
        return True
    else if CNF contains an empty clause then
        return False
    else if CNF contains a pure literal x then
        return DPLL(CNF(x))
    else if CNF contains a unit clause {u} then
        return DPLL(CNF(u))
    else
        choose a variable x that appears in CNF
        if DPLL(CNF(x)) = True then return True
        else return DPLL(CNF(\neg x))
}
```
Stochastic Local Search

Uses **complete assignments** instead of partial
Start with **random state**
Flip variables in unsatisfied clauses
**Hill-climbing:** Minimize # unsatisfied clauses
Avoid **local minima:** Random flips
Multiple restarts

---

The WalkSAT Algorithm

```plaintext
WalkSAT(CNF, max-tries, max-flips, p) {
    for i ← 1 to max-tries do
        solution = random truth assignment
        for j ← 1 to max-flips do
            if all clauses in CNF satisfied then
                return solution
            c ← random unsatisfied clause in CNF
            with probability p
                flip a random variable in c
            else
                flip variable in c that maximizes
                number of satisfied clauses
        return failure
    }
}
```