NP completeness and computational tractability

How do we measure efficiency?

- platform independent, implementation-detail independent \( \rightarrow \) ignore constant factors, use big \( \mathcal{O} \) notation when we talk about running time.

- instance independent \( \rightarrow \) worst-case analysis (sometimes average case analysis)

- of predictive value with respect to increasing input size, tells us how algorithm scales \( \rightarrow \) want to measure rate of growth of \( T(n) \) as function of \( n \), the input size.

Asymptotic, worst-case analysis
Seek polynomial time algorithms

Worst-Case Analysis

Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size \( N \).

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time. Obtain bound on running time of algorithm on random input as a function of input size \( N \).

- Hard (or impossible) to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Polynomial-Time

Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes \( 2^n \) time or worse for inputs of size \( N \).
- Unacceptable in practice.

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor \( C \).

- There exists constants \( c > 0 \) and \( d > 0 \) such that on every input of size \( N \), its running time is bounded by \( c N^d \) steps.

Def. An algorithm is \( \text{poly-time} \) if the above scaling property holds.

Worst-Case Polynomial-Time

Def. An algorithm is \textit{efficient} if its running time is polynomial.

Justification. It really works in practice!

- Although \( 0.02 \times 10^3 \times N^2 \) is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.
Why It Matters

Moore’s Law

The prediction that transistor density and hence the speed of computers will double every 18 months or so.

Based on observation of 1960–1965
Has pretty much held for last 40 years

Does this provide disincentive to develop efficient (polynomial time) algorithms?

NO!!

Running time of alg Max input size 2x speedup 2^10x speedup
in time T

Exponential algorithms make polynomially slow progress, while polynomial algorithms advance exponentially fast!

Asymptotic Order of Growth

Upper bounds. $T(n) = O(f(n))$ if there exist constants $c > 0$ and $n_0$ such that for all $n > n_0$ we have $T(n) \leq c \cdot f(n)$.

Lower bounds. $T(n) = \Omega(f(n))$ if there exist constants $c > 0$ and $n_0$ such that for all $n > n_0$ we have $T(n) \geq c \cdot f(n)$.

Tight bounds. $T(n) = \Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Ex: $T(n) = 32n^2 + 17n + 32$.

$T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.

$T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Omega(n^3)$.

Classify Problems

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

For any nice function $T(n)$
There are problems that require more than $T(n)$ time to solve.

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

NP-completeness. Show that these fundamental 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.** $X \leq_p Y$.

**Remarks.**
- We pay for time to write down instances sent to black box instances of $Y$ must be of polynomial size.
- Note: Cook reducibility.

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 $X \equiv_p Y$.

---

**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 $X \equiv_p Y$.

---

**Reduction By Simple Equivalence**

**Basic reduction strategies.**
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

**Independent Set**

**INDEPENDENT SET.** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| = k$, and for each edge, at least 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.

**Vertex Cover**

**VERTEX COVER.** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| = k$, and for each edge, at least one of its endpoints is in $S$?

**Ex.** Is there a vertex cover of size $4$? Yes.

**Ex.** Is there a vertex cover of size $3$? No.

**Vertex Cover and Independent Set**

Claim. $\text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET}$.

Pf. We show $S$ is an independent set iff $V - S$ is a vertex cover.
Claim. VERTEX-COVER \( \leq_p \) INDEPENDENT-SET.

\( \text{Pf.} \) We show \( S \) is an independent set iff \( V - S \) is a vertex cover.

\[ \begin{align*}
&\Rightarrow \\
&\text{Let } S \text{ be any independent set.} \\
&\text{Consider an arbitrary edge } (u, v). \\
&S \text{ independent } \implies u \notin S \text{ or } v \notin S \implies u \in V - S \text{ or } v \in V - S. \\
&\text{Thus, } V - S \text{ covers } (u, v). \\
\end{align*} \]

\[ \begin{align*}
&\Leftarrow \\
&\text{Let } V - S \text{ be any vertex cover.} \\
&\text{Consider two nodes } u \in S \text{ and } v \in S. \\
&\text{Observe that } (u, v) \notin E \text{ since } V - S \text{ is a vertex cover.} \\
&\text{Thus, no two nodes in } S \text{ are joined by an edge } \implies S \text{ independent set.} \\
\end{align*} \]

**Set Cover**

**SET COVER:** Given a set \( U \) of elements, a collection \( S_1, S_2, \ldots, S_m \) of subsets of \( U \), and an integer \( k \), does there exist a collection of \( k \) of these sets whose union is equal to \( U \)?

**Sample application:**
- \( m \) available pieces of software.
- \( U \) of \( n \) capabilities that we would like our system to have.
- The \( i \)th piece of software provides the set \( S_i \subseteq U \) of capabilities.
- Goal: achieve all \( n \) capabilities using fewest pieces of software.

**Ex:**

\[
\begin{align*}
U &= \{1, 2, 3, 4, 5, 6, 7\} \\
k &= 2 \\
S_1 &= \{3, 7\} \\
S_2 &= \{2, 4\} \\
S_3 &= \{3, 4, 5, 6\} \\
S_4 &= \{5\} \\
S_5 &= \{1\} \\
S_6 &= \{1, 2, 6, 7\} \\
\end{align*}
\]

**Polynomial-Time Reduction**

**Basic strategies.**
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

**Reductions via "Gadgets"**

**Basic reduction strategies.**
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."
Satisfiability

Literal: A Boolean variable or its negation.
- \( x_i \) or \( \overline{x_i} \)

Clause: A disjunction of literals.
- \( C_i = x_i \lor \overline{x_i} \)

Conjunctive normal form: A propositional formula 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.

\[ x_1 \lor x_2 \lor x_3 \]

Each corresponds to a different variable.

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 \( k \) if \( \Phi \) 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.

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

Ex: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \).

Self-Reducibility

Decision problem. Does there exist a vertex cover of size \( k \) or \( k+1 \)?

Search problem. Find vertex cover of minimum cardinality.

Self-reducibility. Search problem \( \leq_p \) decision version.
- Applies to all (NP-complete) problems we discuss.
- Justifies our focus on decision problems.

Ex: to find min cardinality vertex cover.
- (Binary) search for cardinality \( k^* \) of min vertex cover.
- Find a vertex \( v \) such that \( G_{\overline{\{v\}}} \) has a vertex cover of size \( k^* - 1 \).
- Any vertex in any min vertex cover will have this property.
- Include \( v \) in the vertex cover.
- Recursively find a min vertex cover in \( G_{\overline{\{v\}}} \).

Definition of NP

Basic reduction strategies.
- Simple equivalence: INDEPENDENT-SET \( \leq_p \) VERTEX-COVER.
- Special case to general case: VERTEX-COVER \( \leq_p \) SET-COVER.
- Encoding with gadgets: 3-SAT \( \leq_p \) INDEPENDENT-SET.

Transitivity. If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).

Pf idea. Compose the two algorithms.

Ex: 3-SAT \( \leq_p \) INDEPENDENT-SET \( \leq_p \) VERTEX-COVER \( \leq_p \) SET-COVER.
### Decision Problems

**Decision problem.**
- X is a set of strings (a language).
- Instance: string s.
- Algorithm A solves problem X: A(s) = yes iff s ∈ 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>Year</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE</td>
<td>Is ( s ) a multiple of ( n )?</td>
<td>Grade school division</td>
<td>51, 17</td>
<td>51, 18</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are ( x ) and ( y ) relatively prime?</td>
<td>Euclid (200 BCE)</td>
<td>34, 29</td>
<td>34, 31</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is ( s ) prime?</td>
<td>AKS (2002)</td>
<td>59</td>
<td>61</td>
</tr>
<tr>
<td>EDIT-DISTANCE</td>
<td>Is the edit distance between ( x ) and ( y ) less than ( k )?</td>
<td>Dynamic programming</td>
<td>55-56</td>
<td>apropriate</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector ( x ) that satisfies ( Ax = b )?</td>
<td>Gaussian elimination</td>
<td>55-56</td>
<td>55-56</td>
</tr>
</tbody>
</table>

### Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer \( s \), is it composite?

**Certificate.** A nontrivial factor \( t \) of \( s \). Note that such a certificate exists if \( s \) is composite. Moreover \( |t| < \sqrt{s} \).

**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 \( 809 \).

**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.

**Example:**

\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (x_4 \lor x_5 \lor x_6) \land (x_5 \lor x_6) \land (x_1 \lor x_5) \land (x_2 \lor x_6) \]

**Instance s**

\[ x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1 \]

**Certificate t**

### 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 ⊆ NP.

Pf. Consider any problem X in P.
   • By definition, there exists a poly-time algorithm A(s) that solves X.
   • Certificate: t = 1, certifier C(s, 1) = A(s).

Claim. NP ⊆ EXP.

Pf. Consider any problem X in NP.
   • By definition, there exists a poly-time certifier C(s, 1) for X.
   • To solve input s, run C(s, 1) on all strings t with |t| ≤ p(|s|).
   • Return yes, if C(s, 1) 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.

Some writers for the Simpsons and Futurama:

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?

we require |y| to be of size polynomial in |x|
we abuse notation
p and blur distinction

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. If P = NP then Y can be solved in poly-time since Y is in NP.

Pf. 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?

The "First" NP-Complete Problem

Theorem. 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 \( C(s, t) = \text{yes} \).

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.
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 \text{\#} Y.

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that X \text{\#} Y then Y is NP-complete.

Pf. Let W be any problem in NP. Then W \text{\#} X \text{\#} Y.

By transitivity, W \text{\#} Y.

Hence Y is NP-complete.

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

Pf. Suffices to show that CIRC-SAT \text{\#} 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:

\[ \begin{align*}
& x_2 = \neg x_3 + x_4 \\
& x_1 = x_4 + x_5 \\
& x_0 = x_1 + x_2 + x_5
\end{align*} \]

Hard-coded input values and output value.

Final step: turn clauses of length \(< 3\) into clauses of length exactly 3.

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.

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

Notable exceptions. Factoring, graph isomorphism.

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”
- Bread applicability and classification power.
- “Captures vast domain of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasible.”

NP-completeness can guide scientific inquiry.

- 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: Nash equilibrium that maximizes social welfare.

Genomics: phylogeny reconstruction.

Molecular engineering: structure of turbulence in shear 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.