Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.
- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Reductions. (Chapter 8)
- Special Cases (Chapter 10)
- Approximations. (Chapter 11)
- Randomization. (Chapter 13)

Algorithm design anti-pattern.
- NP-completeness. (Chapter 8)  \( O(n^k) \) algorithm unlikely.
Classify Problems According to Computational Requirements

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


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

Desiderata. Classify problems according to minimum time to solve.

Provably requires polynomial time.
- Every problem we've seen so far! (The algorithm is the proof)

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

Frustrating news. Many fundamental problems have defied classification for decades.

This chapter. Show that these problems are "computationally equivalent", i.e. 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. $X \leq_P Y$.

IMPORTANT: We pay for time to write down instances sent to oracle, so instances of Y must be of polynomial size.
Purpose of Polynomial-Time Reductions

**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 $X \equiv_P Y$.

---

Decision Problems

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

**More intuitively.**
- A decision problem is one that asks a yes/no question. e.g.
  - is there a path of length \( \leq k \)?
  - can we fill our knapsack with items of total value \( \geq v \)?
  - is there an RNA sequence with \( \geq p \) pairs?

**Will come back to show why these are useful questions.**
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 $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

**Ex.** Is there a set $S$ of size $\geq 6$?
**Ex.** Of size $\geq 7$?
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 $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$?

**Ex.** Is there a set $S$ of size $\leq 4$?
**Ex.** Of size $\leq 3$?

Vertex Cover and Independent Set

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

**Pf.** We show $S$ is an independent set iff $V - S$ is a vertex cover.
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.

\( \Rightarrow \)

- Let \( S \) be any independent set.
- Consider an arbitrary edge \((u, v)\).
- \( S \) independent \( \Rightarrow u \notin S \) or \( v \notin S \) \( \Rightarrow u \in V - S \) or \( v \in V - S \).
- Thus, \( V - S \) covers \((u, v)\).

\( \Leftarrow \)

- Let \( V - S \) be any vertex cover.
- Consider two nodes \( u \in S \) and \( v \in S \).
- Observe that \((u, v) \notin E \) since \( V - S \) is a vertex cover.
- Thus, no two nodes in \( S \) are joined by an edge \( \Rightarrow S \) independent set.

Polynomial-Time Reduction

**Given the equivalence.** \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

**Design algorithms.** Given an algorithm for \( \text{VERTEX-COVER} \), have an algorithm for \( \text{INDEPENDENT-SET} \) using the reduction (and vice versa).

**Establish tractability.** If \( \text{VERTEX-COVER} \) can be solved in polynomial-time, then \( \text{INDEPENDENT-SET} \) can also be solved in polynomial time (and vice versa).

**Establish intractability.** If \( \text{VERTEX-COVER} \) cannot be solved in polynomial-time, then \( \text{INDEPENDENT-SET} \) cannot be solved in polynomial time (and vice versa).
Reduction from Special Case to General Case

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

**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 \( \leq k \) of these sets whose union is equal to \( U \)?

**Sample application.**
- \( M \) potential employees.
- \( U \) of \( N \) skills our company needs.
- The \( i \)th potential employee has the set \( S_i \subseteq U \) of skills.
- **Goal:** get all \( N \) skills hiring fewest number of employees.

**Ex:**

\[
\begin{array}{c}
U = \{1, 2, 3, 4, 5, 6, 7\} \\
k = 2 \\
S_1 = \{3, 7\} \\
S_4 = \{2, 4\} \\
S_2 = \{3, 4, 5, 6\} \\
S_5 = \{5\} \\
S_3 = \{1\} \\
S_6 = \{1, 2, 6, 7\}
\end{array}
\]
**Vertex Cover Reduces to Set Cover**

**Claim.** VERTEX-COVER \(\leq_p\) SET-COVER.

**Pf.** Given a VERTEX-COVER instance \(G = (V, E), k\), we construct a set cover instance whose size equals the size of the vertex cover instance.

**Construction.**
- Create SET-COVER instance:
  - \(k = k\), \(U = E\), \(S_v = \{e \in E : e \text{ incident to } v\}\)
- Set-cover of size \(\leq k\) iff vertex cover of size \(\leq k\).

**Usefulness of Polynomial-Time Reduction**

**Given the reduction:** VERTEX-COVER \(\leq_p\) SET-COVER.

**Design Algorithms:** Given an algorithm for SET-COVER, we have an algorithm for VERTEX-COVER using the reduction (but not vice versa).

**Establish tractability:** If SET-COVER can be solved in polynomial-time, then VERTEX-COVER can also be solved in polynomial time (but not necessarily vice versa).

**Establish intractability.** If VERTEX-COVER cannot be solved in polynomial-time, then SET-COVER cannot be solved in polynomial time (but not necessarily vice versa).
8.2 Reductions via "Gadgets"

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

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

Clause: A disjunction of literals. \( C_j = x_1 \vee \overline{x_2} \vee x_3 \)

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 distinct literals.

Ex: \( (\overline{x_1} \vee x_2 \vee x_3) \land (x_1 \vee \overline{x_2} \vee x_3) \land (x_2 \vee x_3) \land (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \)

Yes: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \).
3 Satisfiability Reduces to Independent Set

**Claim.** $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$.  

**Pf.** Given an instance $\Phi$ of $3\text{-SAT}$, construct an instance $(G, k)$ of $\text{INDEPENDENT-SET}$ with 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.

$k = 3$

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

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.
- Truth assignment is consistent and all clauses are satisfied.

$k = 3$

$$\Phi = \overline{x_1} \lor x_2 \lor x_3 \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$
3 Satisfiability Reduces to Independent Set

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

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

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

Polynomial-Time Reduction

Established one-sided reduction. \( 3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \)

Design Algorithms. Given an algorithm for \( \text{INDEPENDENT-SET} \), we have an algorithm for \( 3\text{-SAT} \) using the reduction (but not vice versa).

Establish tractability. If \( \text{INDEPENDENT-SET} \) can be solved in polynomial-time, then \( 3\text{-SAT} \) can also be solved in polynomial time (but not necessarily vice versa).

Establish intractability. If \( 3\text{-SAT} \) cannot be solved in polynomial-time, then \( \text{INDEPENDENT-SET} \) cannot be solved in polynomial time (but not necessarily vice versa).
Review

**Basic reduction strategies.**
- Simple equivalence: $\text{INDEPENDENT-SET} \equiv P \text{VERTEX-COVER}$.
- Special case to general case: $\text{VERTEX-COVER} \leq P \text{SET-COVER}$.
- Encoding with gadgets: $\text{3-SAT} \leq P \text{INDEPENDENT-SET}$.

**Transitivity.** If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

*Pf idea.* Compose the two algorithms.

**Ex:** $3\text{-SAT} \leq_P \text{INDEPENDENT-SET} \leq_P \text{VERTEX-COVER} \leq_P \text{SET-COVER}$
So, $3\text{-SAT} \leq_P \text{SET-COVER}$.

---

**In-Class Exercise**

Determine if the following "proofs" hold.
- We saw a pseudo-polynomial time algorithm for solving the knapsack problem, and showed the subset sum problem was a special case. Thus, the best algorithm for subset sum is pseudo-polynomial.
- If $W \leq_P X, X \leq_P Y, Y \leq_P Z, \text{ and } Z \leq_P W$, then by transitivity we know that $X \equiv P Y$.
- We saw an algorithm in class for finding the closest pair of points in 2D. This is a special case of finding the closest pairs of points in 3D, i.e. $2\text{D-POINTS} \leq_P 3\text{D-POINTS}$. Thus, there exists a polynomial time algorithm for the 3D case.
- Given a polynomial time algorithm that solves the optimization Min Spanning Tree problem, there is a polynomial time algorithm that solves the decision MST problem (i.e. is there a MST of size $< k$).
Self-Reducibility: Not in book!!

**Decision problem.** Does there exist a vertex cover of size \( \leq k \)?

**Optimization problem.** Find vertex cover of minimum cardinality.

**Self-reducibility.** Optimization problem \( X \leq_P \) decision problem \( X \).
- Applies to all NP-complete problems.
- 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 \( G - \{v\} \) has a vertex cover of size \( \leq k^* - 1 \).
  - any vertex in any min vertex cover will have this property
- Include \( v \) in the vertex cover.
- Recursively find a min vertex cover in \( G - \{v\} \).
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.
**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>$\begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 0 &amp; 0 \ 1 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \ 0 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

**Definition of NP**

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

"certificate" or "witness"

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

$C(s, t)$ is a poly-time algorithm and $|t| = p(|s|)$ for some polynomial $p()$.

**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.** \( s = 437,669 \).

\( 437,669 = 541 \times 809 \)

**Certificate.** \( t = 541 \) or \( 809 \).

**Conclusion.** COMPOSITES is in NP.

Certifiers and Certificates: 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.**

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

**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: $t = \text{dummy}$, 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.

The Simpson's: P = NP?

Copyright © 1990, Matt Groening
Futurama: $P = NP$?

Looking for a Job?

Some writers for the Simpsons and Futurama.
8.4 NP-Completeness

**NP-complete.** A problem such that
- \( Y \in \text{NP} \)
- for every problem \( X \) in \( \text{NP} \), \( X \leq_p Y \).

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

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

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

**Remark.** The condition that \( Y \in \text{NP} \) is important.

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

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?

$$\text{output}$$

$$\text{inputs}$$

$$\text{hard-coded inputs}$$

yes: 1 0 1 1 0
The "First" NP-Complete Problem

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

**Pf.** (sketch)

Any algorithm that takes $n$ bits as input and produces a yes/no answer can be represented by such a circuit.

If algorithm takes poly-time, then the circuit is of poly-size.

---

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:
  - $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \lor x_3$, $\neg x_2 \lor \neg x_3$
  - $x_4 = x_4 \lor x_5 \Rightarrow$ add 3 clauses: $x_4 \lor \neg x_5$, $x_4 \lor x_5$, $\neg x_4 \lor x_5$
  - $x_0 = x_1 \land x_2 \Rightarrow$ add 3 clauses: $x_0 \lor x_1$, $\neg x_0 \lor x_2$, $x_0 \lor \neg x_1 \lor \neg x_2$

- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: $x_0$
  - $x_0 = 1 \Rightarrow$ add 1 clause:

- Final step: turn clauses of length $< 3$ into clauses of length exactly 3.  ■
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."

Some 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.