Algorithm design patterns and antipatterns

Algorithm design patterns.
- Greedy.
- Divide and conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Algorithm design antipatterns.
- NP-completeness. \(O(n^k)\) algorithm unlikely.
- PSPACE-completeness. \(O(n^k)\) certification algorithm unlikely.
- Undecidability. No algorithm possible.

Classify problems according to computational requirements

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

A working definition. Those with poly-time algorithms.

- Turing machine, word RAM, uniform circuits, ...
- constants tend to be small, e.g., \(3 n^2\)

Theory. Definition is broad and robust.

Practice. Poly-time algorithms scale to huge problems.
Classify problems according to computational requirements

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

**A working definition.** Those with poly-time algorithms.

<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>min cut</td>
<td>max cut</td>
</tr>
<tr>
<td>2-satisfiability</td>
<td>3-satisfiability</td>
</tr>
<tr>
<td>planar 4-colorability</td>
<td>planar 3-colorability</td>
</tr>
<tr>
<td>bipartite vertex cover</td>
<td>vertex cover</td>
</tr>
<tr>
<td>matching</td>
<td>3d-matching</td>
</tr>
<tr>
<td>primality testing</td>
<td>factoring</td>
</tr>
<tr>
<td>linear programming</td>
<td>integer linear programming</td>
</tr>
</tbody>
</table>

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

Provably requires exponential time.
- Given a constant-size program, does it halt in at most \(k\) steps?
- Given a board position in an \(n\)-by-\(n\) generalization of checkers, can black guarantee a win?

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

Poly-time reductions

**Desiderata’**. Suppose we could solve problem \(Y\) in polynomial time. What else could we solve in polynomial time?

**Reduction.** Problem \(X\) polynomial-time (Cook) 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 \(Y\).

Computational model supplemented by special piece of hardware that solves instances of \(Y\) in a single step

\[
\text{input size} = c + \log k
\]

Using forced capture rule

\[
\text{instance } I \\
(\text{of } X)
\]

Algorithm for \(X\)

Algorithm for \(Y\)

Solution \(S\) to \(I\)

\(X \leq_p Y\)

Notation. We pay for time to write down instances of \(Y\) sent to oracle \(\Rightarrow\) instances of \(Y\) must be of polynomial size.

**Novice mistake.** Confusing \(X \leq_p Y\) with \(Y \leq_p X\).
Suppose that $X \leq_p Y$. Which of the following can we infer?

A. If $X$ can be solved in polynomial time, then so can $Y$.

B. $X$ can be solved in poly time iff $Y$ can be solved in poly time.

C. If $X$ cannot be solved in polynomial time, then neither can $Y$.

D. If $Y$ cannot be solved in polynomial time, then neither can $X$.

Poly-time reductions

**Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial time, then $X$ can 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 both $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$. In this case, $X$ can be solved in polynomial time iff $Y$ can be.

Bottom line. Reductions classify problems according to relative difficulty.

Which of the following poly-time reductions are known?

A. $\text{FIND-MAX-FLOW} \leq_p \text{FIND-MIN-CUT}$.

B. $\text{FIND-MIN-CUT} \leq_p \text{FIND-MAX-FLOW}$.

C. Both A and B.

D. Neither A nor B.

8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Independent set

**INDEPENDENT-SET.** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or more) vertices such that no two are adjacent?

**Ex.** Is there an independent set of size $\geq 6$?
**Ex.** Is there an independent set of size $\geq 7$?

---

Vertex cover

**VERTEX-COVER.** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

**Ex.** Is there a vertex cover of size $\leq 4$?
**Ex.** Is there a vertex cover of size $\leq 3$?

---

**Vertex cover and independent set reduce to one another**

**Theorem.** \textsc{Independent-Set} $\equiv_{P}$ \textsc{Vertex-Cover}.

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

---

**Intractability: quiz 3**

**Consider the following graph $G$. Which are true?**

**A.** The white vertices are a vertex cover of size 7.
**B.** The black vertices are an independent set of size 3.
**C.** Both A and B.
**D.** Neither A nor B.

---
Vertex cover and independent set reduce to one another

Theorem. INDEPENDENT-SET \cong_p VERTEX-COVER.

Pf. We show \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).

\[
\begin{align*}
\text{• Let } S \text{ be any independent set of size } k. \\
\text{• } V - S \text{ is of size } n - k. \\
\text{• Consider an arbitrary edge } (u, v) \in E. \\
\text{• } S \text{ independent } \Rightarrow \text{ either } u \notin S, \text{ or } v \notin S, \text{ or both.} \\
\quad \quad \Rightarrow \text{ either } u \in V - S, \text{ or } v \in V - S, \text{ or both.} \\
\text{• Thus, } V - S \text{ covers } (u, v). \quad \blacksquare
\end{align*}
\]

Set cover

SET-COVER. Given a set \( U \) of elements, a collection \( S \) of subsets of \( U \), and an integer \( k \), are there \( \leq k \) of these subsets whose union is equal to \( U \)?

Sample application.

\[
\begin{align*}
\text{• } m \text{ available pieces of software.} \\
\text{• Set } U \text{ of } n \text{ capabilities that we would like our system to have.} \\
\text{• The } i^\text{th} \text{ piece of software provides the set } S_i \subseteq U \text{ of capabilities.} \\
\text{• Goal: achieve all } n \text{ capabilities using fewest pieces of software.}
\end{align*}
\]

Intractability: quiz 4

Given the universe \( U = \{ 1, 2, 3, 4, 5, 6, 7 \} \) and the following sets, which is the minimum size of a set cover?

\[
\begin{align*}
\text{A.} & \quad 1 & \quad U = \{ 1, 2, 3, 4, 5, 6, 7 \} \\
\text{B.} & \quad 2 & \quad S_{a} = \{ 1, 4, 6 \} \quad S_{b} = \{ 1, 6, 7 \} \\
\text{C.} & \quad 3 & \quad S_{c} = \{ 2, 3, 6 \} \quad S_{d} = \{ 1, 3, 5, 7 \} \\
\text{D.} & \quad \text{None of the above.} & \quad S_{e} = \{ 2, 6, 7 \} \quad S_{f} = \{ 3, 4, 5 \}
\end{align*}
\]
Vertex cover reduces to set cover

**Theorem.** \( \text{VERTEX-COVER} \leq_p \text{SET-COVER}. \)

**Pf.** Given a \( \text{VERTEX-COVER} \) instance \( G = (V, E) \) and \( k \), we construct a \( \text{SET-COVER} \) instance \( (U, S, k) \) that has a set cover of size \( k \) iff \( G \) has a vertex cover of size \( k \).

**Construction.**
- Universe \( U = E \).
- Include one subset for each node \( v \in V \): \( S_v = \{ e \in E : e \text{ incident to } v \} \).

\[
\begin{align*}
U &= \{ 1, 2, 3, 4, 5, 6, 7 \} \\
S_a &= \{ 3, 7 \} \\
S_b &= \{ 2, 4 \} \\
S_c &= \{ 3, 4, 5, 6 \} \\
S_d &= \{ 5 \} \\
S_e &= \{ 1 \} \\
S_f &= \{ 1, 2, 6, 7 \}
\end{align*}
\]

**Lemma.** \( G = (V, E) \) contains a vertex cover of size \( k \) iff \( (U, S, k) \) contains a set cover of size \( k \).

**Pf.** Let \( Y \subseteq V \) be a vertex cover of size \( k \) in \( G \).
- Then \( Y = \{ S_v : v \in Y \} \) is a set cover of size \( k \).  

8. **INTRACTABILITY I**

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Satisfiability

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

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

**Conjunctive normal form (CNF).** A propositional formula \( \Phi \) that is a conjunction of clauses.

\[ \Phi = C_1 \land C_2 \land C_3 \land C_4 \]

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

**3-SAT.** SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

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

*yes* instance: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}, x_4 = \text{false} \)

**Key application.** Electronic design automation (EDA).

Satisfiability is hard

**Scientific Hypothesis.** There does not exist a poly-time algorithm for 3-SAT.

**P vs. NP.** This hypothesis is equivalent to \( P \neq NP \) conjecture.

3-satisfiability reduces to independent set

**Theorem.** \( 3\text{-SAT} \leq_P \text{INDEPENDENT-SET} \).

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

**Construction.**
- \( G \) contains 3 nodes for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

3-satisfiability reduces to independent set

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

**Pf.** \( \Rightarrow \) Consider any satisfying assignment for \( \Phi \).
- Select one true literal from each clause/triangle.
- This is an independent set of size \( k = |\Phi| \). ■

**Key application.** Electrical design automation (EDA).
3-satisfiability reduces to independent set

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

Pf. \( \iff \) Let \( S \) be independent set of size \( k \).
- \( S \) must contain exactly one node in each triangle.
- Set these literals to true (and remaining literals consistently).
- All clauses in \( \Phi \) are satisfied. \( \blacksquare \)

"no" instances of 3-SAT are solved correctly

\[

g

k = 3

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

Review

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, Search, and Optimization Problems**

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

**Search problem.** Find a vertex cover of size \( \leq k \).

**Optimization problem.** Find a vertex cover of minimum size.

**Goal.** Show that all three problems poly-time reduce to one another.

---

8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Hamilton cycle

**Hamilton-Cycle.** Given an undirected graph $G = (V, E)$, does there exist a cycle $\Gamma$ that visits every node exactly once?

Directed Hamilton cycle reduces to Hamilton cycle

**Directed-Hamilton-Cycle.** Given a directed graph $G = (V, E)$, does there exist a directed cycle $\Gamma$ that visits every node exactly once?

Theorem. **Directed-Hamilton-Cycle** $\leq_p$ **Hamilton-Cycle**.

Pf. Given a directed graph $G = (V, E)$, construct a graph $G'$ with $3n$ nodes.

Directed Hamilton cycle reduces to Hamilton cycle

**Lemma.** $G$ has a directed Hamilton cycle iff $G'$ has a Hamilton cycle.

Pf. $\Rightarrow$

- Suppose $G$ has a directed Hamilton cycle $\Gamma$.
- Then $G'$ has an undirected Hamilton cycle (same order).

Pf. $\Leftarrow$

- Suppose $G'$ has an undirected Hamilton cycle $\Gamma'$.
- $\Gamma'$ must visit nodes in $G'$ using one of following two orders:
  - black, white, blue, black, white, blue, black, white, blue, ...
  - black, blue, white, black, blue, white, black, blue, white, ...
- Black nodes in $\Gamma'$ comprise either a directed Hamilton cycle $\Gamma$ in $G$, or reverse of one.
3-satisfiability reduces to directed Hamilton cycle

**Theorem.** 3-SAT \(\leq_p\) DIRECTED-HAMILTON-CYCLE.

**Pf.** Given an instance \(\Phi\) of 3-SAT, we construct an instance \(G\) of DIRECTED-HAMILTON-CYCLE that has a Hamilton cycle iff \(\Phi\) is satisfiable.

**Construction overview.** Let \(n\) denote the number of variables in \(\Phi\). We will construct a graph \(G\) that has \(2^n\) Hamilton cycles, with each cycle corresponding to one of the \(2^n\) possible truth assignments.

---

**Intractability: quiz 5**

Which is truth assignment corresponding to Hamilton cycle below?

- **A.** \(x_1 = true, x_2 = true, x_3 = true\)
- **B.** \(x_1 = true, x_2 = true, x_3 = false\)
- **C.** \(x_1 = false, x_2 = false, x_3 = true\)
- **D.** \(x_1 = false, x_2 = false, x_3 = false\)
3-satisfiability reduces to directed Hamilton cycle

**Construction.** Given 3-SAT instance \( \Phi \) with \( n \) variables \( x_i \) and \( k \) clauses.
- For each clause: add a node and 2 edges per literal.

\[
\begin{align*}
C_1 &= x_1 \lor \overline{x_2} \lor x_3 \\
C_2 &= \overline{x_1} \lor x_2 \lor \overline{x_3}
\end{align*}
\]

\( \text{clause node 1} \quad \text{clause node 2} \)

3-satisfiability reduces to directed Hamilton cycle

**Lemma.** \( \Phi \) is satisfiable iff \( G \) has a Hamilton cycle.

**Pf.** \( \Rightarrow \)
- Suppose 3-SAT instance \( \Phi \) has satisfying assignment \( x^* \).
- Then, define Hamilton cycle \( \Gamma \) in \( G \) as follows:
  - if \( x_i = \text{true} \), traverse row \( i \) from left to right
  - if \( x_i = \text{false} \), traverse row \( i \) from right to left
  - for each clause \( C_j \), there will be at least one row \( i \) in which we are going in “correct” direction to splice clause node \( C_j \) into cycle (and we splice in \( C_j \) exactly once) \( \blacksquare \)

Poly-time reductions

- 3-Sat, poly-time reduces to Independent-Set
- 3-Sat, poly-time reduces to Set-Cover
- 3-Sat, poly-time reduces to Vertex-Cover
- 3-Sat, poly-time reduces to Hamilton-cycle
- 3-Sat, poly-time reduces to Directional-Hamilton-cycle
- 3-Sat, poly-time reduces to 3-Color
- 3-Sat, poly-time reduces to Subset-Sum

packing and covering sequencing partitioning numerical

constraint satisfaction

INDEPENDENT-SET

3-SAT

DIR-HAM-CYCLE

3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

KNAPSACK

packing and covering sequencing partitioning numerical
8. **INTRACTABILITY I**

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems

---

**My hobby**

My hobby: Embedding NP-complete problems in restaurant orders.

![Restaurant menu]

NP-Complete by Randall Munro
http://xkcd.com/287
Creative Commons Attribution-NonCommercial 2.5

---

**Subset sum**

**SUBSET-SUM.** Given $n$ natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?


**Yes.** $215 + 355 + 355 + 580 = 1505$.

**Remark.** With arithmetic problems, input integers are encoded in binary. Poly-time reduction must be polynomial in binary encoding.

---

**Subset sum**

**Theorem.** $3$-SAT $\leq_P$ SUBSET-SUM.

**Pf.** Given an instance $\Phi$ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff $\Phi$ is satisfiable.
3-satisfiability reduces to subset sum

**Construction.** Given 3-SAT instance $\Phi$ with $n$ variables and $k$ clauses, form $2n + 2k$ decimal integers, each having $n + k$ digits:
- Include one digit for each variable $x_i$ and one digit for each clause $C_j$.
- Include two variables for each variable $x_i$.
- Include two numbers for each clause $C_j$.
- Sum of each $x_i$ digit is 1; sum of each $C_j$ digit is 4.

**Key property.** No carries possible $\Rightarrow$ each digit yields one equation.

**3-Sat instance**

$$
\begin{align*}
C_1 &= \neg x_1 \lor x_2 \lor x_3 \\
C_2 &= x_1 \lor \neg x_2 \lor x_3 \\
C_3 &= \neg x_1 \lor \neg x_2 \lor \neg x_3
\end{align*}
$$

dummies to get clause columns to sum to 4

$\begin{bmatrix}
1 & 1 & 1 & 4 & 4 & 4 & 111,444
\end{bmatrix}$

3-satisfiability reduces to subset sum

**Lemma.** $\Phi$ is satisfiable iff there exists a subset that sums to $W$.

**Pf.** Suppose there exists a subset $S'$ that sums to $W$.
- Digit $x_i$ forces subset $S'$ to select either row $x_i$ or row $\neg x_i$ (but not both).
- If row $x_i$ selected, assign $x_i^* = true$; otherwise, assign $x_i^* = false$.

Digit $C_j$ forces subset $S'$ to select at least one literal in clause. $\Box$

**3-Sat instance**

$$
\begin{align*}
C_1 &= \neg x_1 \lor x_2 \lor x_3 \\
C_2 &= x_1 \lor \neg x_2 \lor x_3 \\
C_3 &= \neg x_1 \lor \neg x_2 \lor \neg x_3
\end{align*}
$$

dummies to get clause columns to sum to 4

$\begin{bmatrix}
1 & 1 & 1 & 4 & 4 & 4 & 111,444
\end{bmatrix}$

Subset sum reduces to knapsack

**Subset sum.** Given $n$ natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

**Knapsack.** Given a set of items $X$, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit $U$, and a target value $V$, is there a subset $S \subseteq X$ such that:

$$
\sum_{i \in S} u_i \leq U, \quad \sum_{i \in S} v_i \geq V
$$

**Recall.** $O(nU)$ dynamic programming algorithm for Knapsack.

**Challenge.** Prove Subset Sum $\leq_P$ Knapsack.

**Pf.** Given instance $(w_1, \ldots, w_n, W)$ of Subset-Sum, create Knapsack instance:
Poly-time reductions

Karp’s 20 poly-time reductions from satisfiability

Dick Karp (1972)
1985 Turing Award

Numerical constraint satisfaction

packing and covering sequencing partitioning numerical