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.

Theory. Definition is broad and robust.

Practice. Poly-time algorithms scale to huge problems.

Turing machine, word RAM, uniform circuits, ...

\(n\) constants tend to be small, e.g., \(3n^2\)
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>

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 $Y$.

input size = $c + \log k$

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 $Y$.

Notation. $X \leq_p Y$.

Note. 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 $Y \leq_p X$. 
Intractability: quiz 1

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 $X$.

Intractability: quiz 2

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.

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.

8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
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.** \( \text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER} \).

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

\[ \Rightarrow \]

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

\[ \Leftarrow \]

• Let \( V - S \) be any vertex cover of size \( n - k \).
• \( S \) is of size \( k \).
• Consider an arbitrary edge \((u, v) \in E\).
• \( V - S \) is a vertex cover \( \Rightarrow \) either \( u \in V - S \), or \( v \in V - S \), or both.
  \[ \Rightarrow \text{either } u \notin S \text{, or } v \notin S \text{, or both.} \]
• Thus, \( S \) is an independent set.

---

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.**

• \( m \) available pieces of software.
• Set \( U \) 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.

**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?

- **A.** 1
- **B.** 2
- **C.** 3
- **D.** None of the above.

\[
\begin{align*}
U &= \{ 1, 2, 3, 4, 5, 6, 7 \} \\
S_a &= \{ 3, 7 \} & S_b &= \{ 2, 4 \} \\
S_c &= \{ 3, 4, 5, 6 \} & S_d &= \{ 5 \} \\
S_e &= \{ 1 \} & S_f &= \{ 1, 2, 6, 7 \} \\
k &= 2
\end{align*}
\]
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 \( v \in V \) \( : \) \( S_v = \{ e \in E : e \text{ incident to } v \} \).

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

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

Pf. \( \Leftarrow \) Let \( Y \subseteq S \) be a set cover of size \( k \) in \( (U, S, k) \).
- Then \( X = \{ v : S_v \in Y \} \) is a vertex cover of size \( k \) in \( G \).

8. INTRACTABILITY

- 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. \( C_j = x_i \lor \overline{x_i} \lor x_j \)

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

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).

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

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 exists 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-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 \( k = |\Phi| \) 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 \( k = |\Phi| \).

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

\( k = 3 \)

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

\( k = 3 \)
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 \textit{true} (and remaining literals consistently).
- All clauses in \( \Phi \) are satisfied. \( \blacksquare \)

\[ G \]

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

\( k = 3 \)

Review

**Basic reduction strategies.**
- Simple equivalence: \textsc{Independent-Set} \( \leq_p \textsc{Vertex-Cover} \).
- Special case to general case: \textsc{Vertex-Cover} \( \leq_p \textsc{Set-Cover} \).
- Encoding with gadgets: 3-SAT \( \leq_p \textsc{Independent-Set} \).

**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 \( \leq_p \textsc{Independent-Set} \leq_p \textsc{Vertex-Cover} \leq_p \textsc{Set-Cover} \).

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:
  - $\ldots, \text{black, white, blue, black, white, blue, black, white, blue,} \ldots$
  - $\ldots, \text{black, blue, white, black, blue, white, black, blue, white,} \ldots$
- 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.** $x_1 = false, x_2 = false, x_3 = false$

---

**Construction.** Given 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.
- Construct $G$ to have $2^n$ Hamilton cycles.
- Intuition: traverse path $i$ from left to right $\iff$ set variable $x_i = true$.

- For each clause: add a node and 2 edges per literal.
- Connect in this way if $x_i$ appears in clause $C_j$.
- Connect in this way if $x_i$ appears in clause $C_k$.

- $x_i = true$
- $x_i = false$
### 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.

![Diagram of construction](image)

- $C_1 = x_1 \lor \overline{x_2} \lor x_3$
- $C_2 = \overline{x_1} \lor x_2 \lor \overline{x_3}$

\[ 3k + 3 \]

### 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 $x_i^* = true$, traverse row $i$ from left to right
  - if $x_i^* = false$, 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)

### Pf. $\Leftarrow$
- Continue in this way, we are left with a Hamilton cycle $\Gamma'$ in $G - \{C_j\}$.
- Set $x_i^* = true$ if $\Gamma'$ traverses row $i$ left-to-right; otherwise, set $x_i^* = false$.
- traversed in “correct” direction, and each clause is satisfied.

### Poly-time reductions

- 3-SAT polytime reduces to Independent Set
- 3-SAT polytime reduces to Directed Hamilton cycle
- 3-SAT polytime reduces to Graph-3-Color
- 3-SAT polytime reduces to Subset Sum
- 3-SAT polytime reduces to Vertex Cover
- 3-SAT polytime reduces to Hamilton Cycle
- 3-SAT polytime reduces to Set Cover

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

---

**Subset sum**

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

**Ex.** \(\{215, 215, 275, 275, 355, 355, 420, 420, 580, 655\}\), \(W = 1505\).
**Yes.** \(215 + 355 + 355 = 1505\).

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

---

**My hobby**

**My hobby: Embedding NP-complete problems in Restaurant orders**

---

**Subset sum**

**Theorem.** \(3\text{-SAT} \leq_p \text{SUBSET-SUM.}\)

**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 digits 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.

```
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
```

3-SAT instance
dummies to get clause columns to sum to 4

$x_1$  1  0  0  0  1  0  100,010
$\neg x_1$  1  0  0  1  0  1  100,101
$x_2$  0  1  0  1  0  0  10,100
$\neg x_2$  0  1  0  0  1  1  10,011
$x_3$  0  0  1  1  1  0  1,110
$\neg x_3$  0  0  1  0  0  1  1,001

Cost: 111,444

```
1 1 4 4 4
```

SUBSET-SUM instance

3-satisfiability reduces to subset sum

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

**Pf.** $\Rightarrow$ 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.

```
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
```

3-SAT instance
dummies to get clause columns to sum to 4

$x_1$  1  0  0  0  1  0  100,010
$\neg x_1$  1  0  0  1  0  1  100,101
$x_2$  0  1  0  1  0  0  10,100
$\neg x_2$  0  1  0  0  1  1  10,011
$x_3$  0  0  1  1  1  0  1,110
$\neg x_3$  0  0  1  0  0  1  1,001

Cost: 111,444

```
1 1 4 4 4
```

SUBSET-SUM instance

SUBSET SUM REDUCES TO KNAPSACK

**SUBSET-SUM.** Given a set $X$, values $u_i \geq 0$, and an integer $U$, is there a subset $S \subseteq X$ whose elements sum to exactly $U$?

**KNAPSACK.** Given a set $X$, 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:

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

**Theorem.** SUBSET-SUM $\leq_P$ KNAPSACK.

**Pf.** Given instance $(w_1, \ldots, w_n, W)$ of SUBSET-SUM, create KNAPSACK instance:
Poly-time reductions

constraint satisfaction

3-Sat

INDEPENDENT-SET

DIR-HAM-CYCLE

3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

KNAPSACK

packing and covering

sequencing

partitioning

numerical

Karp’s 21 poly-time reductions from satisfiability

Dick Karp (1972)

1985 Turing Award

FIGURE 1: Complete Problem