8. NP and Computational Intractability

Algorithmic Design Patterns and Anti-Patterns

Algorithmic design patterns.
- Greed. \( O(n \log n) \) interval scheduling.
- Divide-and-conquer. \( O(n \log n) \) sorting.
- Dynamic programming. \( O(n^2) \) edit distance.
- Network flow. \( O(n^3) \) bipartite matching.

Algorithmic anti-design patterns.
- NP-completeness \( O(n^k) \) algorithm unlikely.
- PSPACE-completeness. \( O(n^k) \) certifying algorithm unlikely.
- Undecidability. No algorithm possible.

NP

We'll look only at decision problems
- Problems for which the answer is yes or no
- Optimization problem: what is the longest path?
- Decision problem: is there a path of size \( k \) or larger?

\( \text{NP} = \) Set of problems for which a solution can be verified in polynomial time
- Given a certificate for a solution (certificate is polynomial in the input size), certifier checks to see whether the solution is correct
- For example, a certificate for path of size \( k \) or larger is the path itself
  - Certifier: check that each edge is in the original graph, check that the path has at least \( k \) edges

NP problems can be solved in exponential time
- Run the certifier on each of the exponential number of certificates

Historical Note

NP stands for Non-deterministic Polynomial time
- Alternative definition of NP: guess a certificate (non-deterministically) and then verify it in polynomial time
- Our definition doesn't care where the certificate comes from: just concentrates on the existence of a polynomial-time certifier

\( \text{P (Polynomial time)} = \) Set of problems for which a solution can be found in polynomial time

Biggest question in Computer Science: Does \( \text{P} = \text{NP} \)?
NP Problems

- NP-hard: as hard as any problem in NP
  - But need not be in NP itself
- NP-complete (NPC): the hardest of the NP problems
  - NP-hard and in NP itself
  - If any NP-complete problem can be solved in polynomial time, all problems in NP can be solved in polynomial time
  - Alternatively: if some NP problem can’t be solved in polynomial time, then no NP-complete problem can be solved in polynomial time

Relationship between P and NP

- \( P \subseteq NP \)

Possibility 1: \( P = NP = NPC \)
- Every NP problem can be solved in polynomial time

Possibility 2: \( P \neq NP \)
- No NP-complete problem can be solved in polynomial time
- Believed by many to be true

Classify Problems According to Computational Requirements

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

A working definition. [Cobham 1960, Edmonds, 1962] **Those with polynomial-time algorithms.**

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

Unfortunately... huge number of fundamental problems have defied classification for decades.

Fortunately... they were shown to be "computationally equivalent" and intractable for all practical purposes.

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.

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

Showing a Problem Q is NP-Complete

Choose a known NP-complete problem, P
Reduce P to Q
- $P \leq_p Q$
  - Prove reduction is correct
  - Prove reduction works in polynomial time
Prove that Q is in NP
Polynomial-Time Reduction

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

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 most one of its endpoints is in $S$?

Ex. Is there an independent set of size $\geq 6$? Yes.
Ex. Is there an independent set of size $\geq 7$? No.

INDEPENDENT SET

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 vertex cover of size $\leq 4$? Yes.
Ex. Is there a vertex cover of size $\leq 3$? No.

Vertex Cover

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

Ex:
- \( U = \{1, 2, 3, \ldots, 12\}, k = 3 \)
- \( S_1 = \{1, 2, 3, 4, 5, 6\} \quad S_2 = \{5, 6, 8, 9\} \)
- \( S_3 = \{1, 4, 7, 10\} \quad S_4 = \{2, 5, 7, 8, 11\} \)
- \( S_5 = \{3, 6, 9, 12\} \quad S_6 = \{10, 11\} \)

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

**INTEGER-PROGRAMMING**: Given rational numbers $a_{ij}, b$ find integers $x_j$ that satisfy:

\[
\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad 1 \leq i \leq m
\]
\[
x_j \geq 0 \quad 1 \leq j \leq n
\]
\[
x_j \text{ integral} \quad 1 \leq j \leq n
\]

**Claim.** VERTEX-COVER $\leq_p$ INTEGER-PROGRAMMING.

\[
\sum_{u \in V} x_u \geq k
\]
\[
x_u + x_{\overline{u}} \geq 1 \quad (u, v) \in E
\]
\[
x_u \geq 0 \quad u \in V
\]
\[
x_u \text{ integral} \quad u \in V
\]

**Polynomial-Time Reduction**

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

**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:** A propositional formula $\Phi$ that is the conjunction of clauses. $\Phi = C_1 \land C_2 \land C_3 \land C_4$

**SAT:** Given propositional formula in conjunctive normal form, does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause is of length exactly 3.

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

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

**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.
3 Satisfiability Reduces to Independent Set

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

\( \Rightarrow \) Let \( S \) be independent set of size \( k \).
- \( S \) must contain exactly one vertex in each triangle.
- Set these literals to true. \( \iff \) and any other variables in a consistent way
- No conflicts.

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

\( \Phi = \{ \overline{x}_1 \lor x_2 \lor x_3 \} \land \{ x_1 \lor \overline{x}_2 \lor \overline{x}_3 \} \land \{ \overline{x}_1 \lor x_2 \lor x_4 \} \)

Self-Reducibility

Decision problem. Does \( G \) have a vertex cover of size \( \leq k \)?

Search problem. Find vertex cover of minimum cardinality.

Self-reducibility. Search problem \( \leq_p \) decision version.
- Applies to all problems in this chapter.
- Justifies our emphasis 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 \( G - \{ v \} \).

8.4 Sequencing Problems

Basic genres.
- 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: 3-COLOR, 3D-MATCHING.
- Numerical problems: SUBSET-SUM, KNAPSACK.
**Hamiltonian Cycle**

**HAM-CYCLE:** given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that contains every vertex in $V$.

**YES:** vertices and edges of a dodecahedron.

**NO:** bipartite graph with odd number of nodes.

**Directed Hamiltonian Cycle**

**DIR-HAM-CYCLE:** given a digraph $G = (V, E)$, does there exist a simple directed cycle $C$ that contains every vertex in $V$?

**Claim.** $\text{DIR-HAM-CYCLE} \leq \text{HAM-CYCLE}$.

**Proof.** Given a directed graph $G = (V, E)$, construct an undirected graph $G'$ with $3n$ vertices.

**Claim.** $G$ has a Hamiltonian cycle if and only if $G'$ does.

**Proof.**

- Suppose $G$ has a directed Hamiltonian cycle $C$.
- Then $G'$ has an undirected Hamiltonian cycle (same order).

- Suppose $G'$ has an undirected Hamiltonian cycle $C'$.
- $C'$ must visit nodes in $G'$ using one of following two orders:
  - $\ldots, G, R, B, G, R, B, G, R, B, \ldots$
  - $\ldots, G, B, R, G, B, R, G, B, \ldots$
- Blue nodes in $C'$ make up directed Hamiltonian cycle $C$ in $G$, or reverse of one.
3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.** $3$-$\text{SAT} \leq_p \text{DIR-HAM-CYCLE}.$

**Proof.** Given an instance $\Phi$ of $3$-$\text{SAT}$, we construct an instance of $\text{DIR-HAM-CYCLE}$ that has a Hamiltonian cycle iff $\Phi$ is satisfiable.

**Construction.** First, create a graph that has $2^n$ Hamiltonian cycles which correspond in a natural way to $2^n$ possible truth assignments.

Given a $3$-$\text{SAT}$ instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.

- For each clause: add a node and 6 edges.

$$C_1 = x_1 \lor x_2 \lor x_3$$

clause node

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

clause node

$\Phi$ is satisfiable iff $G$ has a Hamiltonian cycle.

**Pf.** $\Rightarrow$

- Suppose $3$-$\text{SAT}$ instance has satisfying assignment $x^*$.

- Then, define Hamiltonian cycle in $G$ as follows:
  - if $x^*_i = 1$, traverse row $i$ from left to right
  - if $x^*_i = 0$, 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 node $C_j$ into tour
3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.** \( \Phi \) is satisfiable iff \( G \) has a Hamiltonian cycle.

**Pf.**
- Suppose \( G \) has a Hamiltonian cycle \( \Gamma \).
  - If \( \Gamma \) enters clause node \( C_j \), it must depart on mate edge.
    - thus, nodes immediately before and after \( C_j \) are connected by an edge \( e \) in \( G \)
    - removing \( C_j \) from cycle, and replacing it with edge \( e \) yields Hamiltonian cycle on \( G - \{C_j\} \)
  - Continuing in this way, we are left with Hamiltonian cycle \( \Gamma' \) in \( G - \{C_1, C_2, \ldots, C_k\} \).
- Set \( x^*_i = 1 \) iff \( \Gamma' \) traverses row \( i \) left to right.
- Since \( \Gamma \) visits each clause node \( C_j \), at least one of the paths is traversed in “correct” direction, and each clause is satisfied.

---

8.5 3-Dimensional Matching

**3-Dimensional Matching**

**3D-MATCHING.** Given \( n \) instructors, \( n \) courses, and \( n \) times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Course</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wayne</td>
<td>COS 423</td>
<td>MW 11-12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 423</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 226</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 126</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Tarjan</td>
<td>COS 523</td>
<td>TTh 3-4:20</td>
</tr>
<tr>
<td>Tarjan</td>
<td>COS 423</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Tarjan</td>
<td>COS 423</td>
<td>TTh 3-4:20</td>
</tr>
<tr>
<td>Sedgewick</td>
<td>COS 226</td>
<td>TTh 3-4:20</td>
</tr>
<tr>
<td>Sedgewick</td>
<td>COS 226</td>
<td>MW 11-12:20</td>
</tr>
<tr>
<td>Sedgewick</td>
<td>COS 423</td>
<td>MW 11-12:20</td>
</tr>
</tbody>
</table>
8.6 Graph Coloring

Basic genres:
- 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: 3-COLOR, 3D-MATCHING.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Register Allocation

Register allocation. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables names, edge between u and v if there exists an operation where both u and v are "live" at the same time.

Observation. [Chaitin, 1982] Can solve register allocation problem iff interference graph is k-colorable.

Fact. 3-COLOR \subseteq_p k-REGISTER-ALLOCATION for any constant k \geq 3.

Planar 3-Colorability

PLANAR-3-COLOR. Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

Yes instance.
### Planar 3-Colorability

**PLANAR-3-COLOR.** Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

**Claim.** $3$-COLOR $\leq_P$ PLANAR-3-COLOR.

**Proof sketch:** Given instance of $3$-COLOR, draw graph in plane, letting edges cross if necessary.
- Replace each edge crossing with the following planar gadget $W$.
  - in any 3-coloring of $W$, opposite corners have the same color
  - any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of $W$

### Planarity

**Def.** A graph is **planar** if it can be embedded on the plane (or sphere) in such a way that no two edges cross.

**Applications:** VLSI circuit design, computer graphics.

![Planar graphs and non-planar graphs](image)

### Planar $k$-Colorability

**PLANAR-2-COLOR.** Solvable in linear time.

**PLANAR-3-COLOR.** NP-complete.

**PLANAR-4-COLOR.** Solvable in $O(1)$ time.

**Theorem.** [Appel-Haken, 1976] Every planar map is 4-colorable.
- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.

**False intuition.** If PLANAR-3-COLOR is hard, then so is PLANAR-4-COLOR and PLANAR-5-COLOR.
8.7 Numerical Problems

Subset Sum

Construction. Given 3-SAT instance \( \Phi \) with \( k \) clauses and \( n \) variables, form the following \( 2n + 2k \) integers (in base 10). 

Claim. 3D-matching iff some subset sums to \( W \).

Pf. No carries possible.

\[
\begin{array}{c|ccccc|c}
\times & y & z & C_1 & C_2 & C_3 & 100,010 \\
\hline
x1 & 1 & 0 & 0 & 0 & 1 & 0 \\
- x1 & 1 & 0 & 0 & 1 & 0 & 010 \\
x2 & 0 & 1 & 0 & 1 & 0 & 0 \\
- x2 & 0 & 1 & 0 & 0 & 1 & 011 \\
x3 & 0 & 0 & 1 & 1 & 0 & 110 \\
- x3 & 0 & 0 & 0 & 1 & 1 & 001 \\
\end{array}
\]

2k dummies to get clause columns to sum to 3

\( W = 111,333 \)

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

Pf. Given an instance \( \Phi \) of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff \( \Phi \) is satisfiable.

Scheduling With Release Times

SCHEDULE-RELEASE-TIMES. Given a set of \( n \) jobs with processing time \( t_i \), release time \( r_i \), and deadline \( d_i \), is it possible to schedule all jobs on a single machine such that job \( i \) is processed with a contiguous slot of \( t_i \) time units in the interval \( [r_i, d_i] \)?

Claim. \( \text{SUBSET-SUM} \leq_p \text{SCHEDULE-RELEASE-TIMES} \).

Pf. Given an instance of SUBSET-SUM \( w_1, \ldots, w_n \) and target \( W \), let \( S = 1 + \sum_j w_j \).

- Create \( n \) jobs with processing time \( t_i = w_i \), release time \( r_i = 0 \), and deadline \( d_i = S+1 \).
- Create job 0 with \( t_0 = 1 \), release time \( r_0 = W \), and deadline \( d_0 = W+1 \).

Can schedule jobs 1 to \( n \) anywhere but \([W, W+1]\)
8.9 A Partial Taxonomy of Hard Problems

Polynomial-Time Reductions

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

packing and covering
sequencing
partitioning
numerical

Undecidable Problems

- Given a computer program, does it halt?
  - Program 1
    - \( i = 10 \)
    - \( \text{while } i > 0 \)
    - \( \text{total} = \text{total} + i \)
  - Program 2
    - \( i = 10 \)
    - \( \text{while } i > 0 \)
    - \( \text{total} = \text{total} + i \)
    - \( i = i - 1 \)
Halting Problem

Does it halt?
- \( i = j = k = n = 3 \)
  loop
    exit when \( i^n + j^n = k^n \)
    increment minimum of \( i, j, k, n \)
  end loop
- I have a marvelous proof that this program never halts, but this slide is too small to contain it!

Assume we have an algorithm \( H \) that can decide whether a given program \( A \) halts on input \( I \):
- \( H(A, I) \)
  - Returns true if the program \( A \) halts on input \( I \)
  - Returns false if the program \( A \) doesn't halt on input \( I \)
- Construct \( H' \):
  - \( H'(A) \)
    - if \( H(A, A) \) then
      loop forever
    else
      return
  - What about \( H'(H') \)? If \( H' \) halts on itself, then it doesn't halt on itself. If \( H' \) doesn't halt on itself, then it does halt on itself
By contradiction, no such algorithm \( H \) exists!

Can't decide much useful about a program

Summary

If you are given a problem:
- If it is undecidable, don't bother trying to solve it
- If it is NP-complete, don't bother trying to come up with a polynomial algorithm
  - Try an approximation algorithm
  - Try a simpler version of the problem