NP-completeness and computational intractability

Lecture 28

Course Instructor: Sikder Huq
Algorithm Design Patterns and Anti-Patterns

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

Ex.
- $O(n \log n)$ interval scheduling.
- $O(n \log n)$ mergesort.
- $O(n^2)$ edit distance.
- $O(n^3)$ bipartite matching.

Algorithm design anti-patterns.
- NP-completeness. $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.
8.1 Polynomial-Time Reductions
**Classify Problems According to Computational Requirements**

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


<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>Matching</td>
<td>3D-matching</td>
</tr>
<tr>
<td>Min cut</td>
<td>Max cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar 4-color</td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</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?

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

This chapter. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.
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 $\Rightarrow$ instances of Y must be of polynomial size.
- Note: Cook reducibility.
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: 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 an independent set of size $\geq 6$? Yes.
Ex. Is there an independent set of size $\geq 7$? 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 $|S| \leq k$, 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.
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. \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

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.
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.**
- $m$ 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.

**Ex:**

<table>
<thead>
<tr>
<th>$U$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1, 2, 3, 4, 5, 6, 7}$</td>
<td>${3, 7}$</td>
<td>${3, 4, 5, 6}$</td>
<td>${1}$</td>
<td>${2, 4}$</td>
<td>${5}$</td>
<td>${1, 2, 6, 7}$</td>
</tr>
</tbody>
</table>
Vertex Cover Reduces to Set Cover

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

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

Construction.

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

\[
\begin{align*}
    \text{VERTEX COVER} & : \quad a \quad b \\
    e_7 & \quad e_2 & \quad e_3 & \quad e_4 \\
    e_1 & \quad e_6 & \quad e_5 \\
    \text{SET COVER} & : \quad U = \{1, 2, 3, 4, 5, 6, 7\} \\
    k = 2 & \quad S_a = \{3, 7\} & \quad S_b = \{2, 4\} \\
    S_c = \{3, 4, 5, 6\} & \quad S_d = \{5\} \\
    S_e = \{1\} & \quad S_f = \{1, 2, 6, 7\}
\end{align*}
\]
Q1: Hitting set

**HITTING SET:** 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 subset of $U$ of size $\leq k$ such that $U$ overlaps with each of the sets $S_1, S_2, \ldots, S_m$?

Show that SET COVER polynomial reduces to HITTING SET.
8.2 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. \[ x_i \text{ or } \overline{x_i} \]

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 CNF formula \( \Phi \), does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

\[ \uparrow \]

each corresponds to a different variable

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

Yes: \( x_1 = \text{true}, x_2 = \text{true} \) \( x_3 = \text{false} \).
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 $k$ iff $\Phi$ is satisfiable.
3 Satisfiability Reduces to Independent Set

Claim. \(3\text{-SAT} \leq_p \text{INDEPENDENT-SET}\).

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

\[
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)
\]
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. and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

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

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

Basic reduction strategies.
- Simple equivalence: INDEPENDENT-SET $\equiv_p$ 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.
**Self-Reducibility**

**Decision problem.** Does there exist 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 (NP-complete) problems in this chapter.
- 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 \( G - \{ v \} \).
  \[
  \text{delete } v \text{ and all incident edges}
  \]
Lecture 29
**Q1: Hitting set**

**HITTING SET:** 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 subset of $U$ of size $\leq k$ such that $U$ overlaps with each of the sets $S_1, S_2, \ldots, S_m$?

Show that SET COVER polynomial reduces to HITTING SET.

<table>
<thead>
<tr>
<th>SET COVER</th>
<th>HITTING SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U = {1, 2, 3, 4, 5, 6, 7}$</td>
<td>$U = {S_a, S_c, S_e, S_b, S_d, S_f}$</td>
</tr>
<tr>
<td>$k = 2$</td>
<td>$k = 2$</td>
</tr>
<tr>
<td>$S_a = {3, 7}$</td>
<td>$S_1 = {S_e, S_f}$</td>
</tr>
<tr>
<td>$S_c = {3, 4, 5, 6}$</td>
<td>$S_2 = {S_b, S_f}$</td>
</tr>
<tr>
<td>$S_e = {1}$</td>
<td>$S_3 = {S_a, S_c}$</td>
</tr>
<tr>
<td>$S_d = {5}$</td>
<td>$S_4 = {S_c, S_b}$</td>
</tr>
<tr>
<td>$S_f = {1, 2, 6, 7}$</td>
<td>$S_5 = {S_c, S_d}$</td>
</tr>
<tr>
<td></td>
<td>$S_6 = {S_c, S_f}$</td>
</tr>
<tr>
<td></td>
<td>$S_7 = {S_a, S_f}$</td>
</tr>
</tbody>
</table>
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.

\[ p(|s|) = |s|^8. \]

PRIMES: \( X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots \} \)

Algorithm. [Agrawal-Kayal-Saxena, 2002]
**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>[0 1 1 ; 1 1 1 ; 1 1 1]</td>
<td>[1 0 0 ; 1 1 1 ; 1]</td>
</tr>
</tbody>
</table>
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 \( |t| \leq p(|s|) \) for some polynomial \( p(\cdot) \).

**Remark.** NP stands for **nondeterministic** polynomial-time.
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.

**Ex.**

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

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**: $t = \varepsilon$, 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.

![Diagram showing the relationship between P, NP, EXP, and their implications on $P = NP$.]

**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.
8.4 NP-Completeness
**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.** $\Leftarrow$ If $P = NP$ then Y can be solved in poly-time since Y is in NP.

**Pf.** $\Rightarrow$ 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?
Circuit Satisfiability (The "First" NP-Complete Problem)

**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?

yes: 1 0 1
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.
**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 ttatat</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector x that satisfies Ax = b?</td>
<td>Gauss–Edmonds elimination</td>
<td>![Matrix]</td>
<td>![Matrix]</td>
</tr>
</tbody>
</table>
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 $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

**Remark.** NP stands for nondeterministic polynomial-time.
Observation. All problems below are NP-complete and polynomial reduce to one another!

NP-Completeness

by definition of NP-completeness
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."

NP-completeness can guide scientific inquiry.
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
Thanks for attending!