COP 4531
Complexity & Analysis of Data Structures & Algorithms

Lecture 18
Reductions and NP-completeness

Thanks to Kevin Wayne and the text authors who contributed to these slides
Classify Problems According to Computational Requirements

- Q. Which problems will we be able to solve in practice?
- A working definition: those with polynomial-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>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

• 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?

• Cannot be solved at all: Given an arbitrary Turing machine T and an input x, Does T halt on x.

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

• But, we can show that many of these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.
Polynomial-Time Reduction

• Suppose we could solve B in polynomial-time. What else could we solve in polynomial time?

• Reduction. Problem A polynomial reduces to problem B if arbitrary instances of problem A can be solved using:
  - Polynomial transform an instance \( a \) of A to a problem \( \beta \) of B
  - The answers are the same. The answer for \( a \) is “yes” if and only if the answer for \( \beta \) is “yes”

• Notation. \( A \leq_p B \).
Polynomial-Time Reduction

• Purpose. Classify problems according to relative difficulty.

• Design algorithms. If \( A \leq_p B \) and \( B \) can be solved in polynomial-time, then \( A \) can also be solved in polynomial time.

• Establish intractability. If \( A \leq_p B \) and \( A \) cannot be solved in polynomial-time, then \( B \) cannot be solved in polynomial time.

• Establish equivalence. If \( A \leq_p B \) and \( B \leq_p A \), we use notation \( A \equiv_p B \).

\[ \text{up to cost of reduction} \]
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 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. \textsc{INDEPENDENT-SET} \equiv_p \textsc{VERTEX-COVER} \\

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

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

$\iff$
- 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 $\implies S$ 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>$U$ = {1, 2, 3, 4, 5, 6, 7}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 2$</td>
</tr>
<tr>
<td>$S_1 = {3, 7}$</td>
</tr>
<tr>
<td>$S_2 = {3, 4, 5, 6}$</td>
</tr>
<tr>
<td>$S_3 = {1}$</td>
</tr>
<tr>
<td>$S_4 = {2, 4}$</td>
</tr>
<tr>
<td>$S_5 = {5}$</td>
</tr>
<tr>
<td>$S_6 = {1, 2, 6, 7}$</td>
</tr>
</tbody>
</table>
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:
    - $k = k$, $U = E$, $S_v = \{e \in E : e$ incident to $v$ $\}$
  - Set-cover of size $\leq k$ iff vertex cover of size $\leq k$. □

Vertex Cover

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$k = 2$

$S_a = \{3, 7\}$
$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$
$S_d = \{5\}$
$S_e = \{1\}$
$S_f = \{1, 2, 6, 7\}$
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. \( x_i \) 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.

\[
\begin{align*}
\text{Ex:} & \quad (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x_1} \lor \overline{x_3} \lor \overline{x_4}) \\
\text{Yes:} & \quad x_1 = \text{true}, \ x_2 = \text{true} \ x_3 = \text{false}.
\end{align*}
\]
3 Satisfiability Reduces to Independent Set

- **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.

- **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.

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

$k = 3$
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: \( \text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER} \).
  - Special case to general case: \( \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).
  - Encoding with gadgets: \( 3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \).

- Transitivity. If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).
- Pf idea. Compose the two algorithms.

- Ex: \( 3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).
Decision Problems vs. Optimization Problems

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

• Optimization problem. Find vertex cover of minimum cardinality.

• It is easy to see that decision problem \( \leq_p \) optimization version.
  - Solve optimization problem; compare solution with \( k \).
  - Since we will generally be interested in proving that a problem is hard, if we can show the decision problem is hard, then we know the corresponding optimization problem is also hard.

• Self-reducibility. Optimization problem \( \leq_p \) decision version.
  - Applies to all (NP-complete) problems we consider.
  - Further justifies our focus on decision problems.
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.

- $PRIMES: X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots\}$
- Algorithm. [2002] $p(|s|) = |s|^8$. 

\[ \text{length of } s \]


# 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>acgggttttta</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] , [4]</td>
<td>[1 0 0] , [1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[2 4 -2] , [2]</td>
<td>[1 1 1] , [1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[0 3 15] , [36]</td>
<td>[0 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: 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| ≤ |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.

- **Certificate.** t = 541 or 809.  
  
  437,669 = 541 × 809

- **Conclusion.** COMPOSITES is in NP.
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.**

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

  instance $s$

  $x_1 = 1, \ x_2 = 1, \ x_3 = 0, \ x_4 = 1$

- **Conclusion.** SAT is in NP. certificate $\vdash$
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: separate \( t \) not needed, 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.
NP-Complete

- NP-complete. A problem $Y$ in NP with the property that for every problem $X$ in NP, $X \leq_p Y$. (Hardest problems in NP)

- **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 $P = NP$. □
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?

![Circuit Diagram]

- Yes: 1 0 1

![Output Diagram]
The "First" NP-Complete Problem

• Theorem. CIRCUIT-SAT is NP-complete.

• Pf. (sketch)
  - Any algorithm that takes a fixed number of bits \( n \) as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

  sketchy part of proof; fixing the number of bits is important, and reflects basic distinction between algorithms and circuits

- Consider some problem \( X \) in NP. It has a poly-time certifier \( C(s, t) \).
  To determine whether \( s \) is in \( X \), need to know if there exists a certificate \( t \) of length \( p(|s|) \) such that \( C(s, t) = \text{yes} \).

- View \( C(s, t) \) as an algorithm on \( |s| + p(|s|) \) bits (input \( s \), certificate \( t \)) and convert it into a poly-size circuit \( K \).
  - first \( |s| \) bits are hard-coded with \( s \)
  - remaining \( p(|s|) \) bits represent bits of \( t \)

- Circuit \( K \) is satisfiable iff \( C(s, t) = \text{yes} \).
Example

- Ex. Construction below creates a circuit $K$ whose inputs can be set so that $K$ outputs true iff graph $G$ has an independent set of size 2.

$G = (V, E), n = 3$

\[ \binom{n}{2} \] hard-coded inputs (graph description)  \hspace{1cm} n \text{ inputs (nodes in independent set)}

Example Construction below creates a circuit $K$ whose inputs can be set so that $K$ outputs true iff graph $G$ has an independent set of size 2.
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. □
• Theorem. 3-SAT is NP-complete.

• Pf. Suffices to show that CIRCUIT-SAT $\leq_p$ 3-SAT since 3-SAT is in NP.

• Hence we now have
  - 3-SAT $\leq_p$ Independent Set $\leq_p$ Vertex Cover $\leq_p$ Set Cover
  - In our NP-Complete Bank
X problems?


 Couldn’t find a poly-time solution bossss 😞?
X problems?


Couldn’t find a poly-time solution boss because none exists.
X problems?


Couldn’t find a poly-time solution boss but neither could all these smart people...
NP-Completeness

- Observation. All problems below are NP-complete and polynomial reduce to one another!

Diagram:
- CIRCUIT-SAT
  - 3-SAT
    - 3-SAT reduces to INDEPENDENT SET
      - INDEPENDENT SET
        - VERTEX COVER
          - SET COVER
    - HAM-CYCLE
      - TSP
    - PLANAR 3-COLOR
      - SCHEDULING
  - GRAPH 3-COLOR
  - SUBSET-SUM

by definition of NP-completeness
Some NP-Complete Problems

- Six basic genres of NP-complete problems and paradigmatic examples.
  - 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: 3D-MATCHING, 3-COLOR.
  - Numerical problems: SUBSET-SUM, KNAPSACK.

- Practice. Most NP problems are either known to be in P or NP-complete.

- Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.