Review: Polynomial-Time Reduction

▶ \( Y \leq_P X \): Problem \( Y \) is polynomial-time reducible to Problem \( X \),

\[
\text{solve} Y(y\text{Input})
\]

Construct \( x\text{Input} \) // poly-time

\[
\text{foo} = \text{solve} X(x\text{Input}) \quad // \text{poly \# of calls}
\]

return yes/no based on foo // poly-time

▶ ... if any instance of Problem \( Y \) can be solved using

1. A polynomial number of standard computational steps
2. A polynomial number of calls to a black box that solves problem \( X \)

▶ Statement about relative hardness

1. If \( Y \leq_P X \) and \( X \in P \), then \( Y \in P \)
2. If \( Y \leq_P X \) and \( Y \not\in P \) then \( X \not\in P \)

Reduction Strategies

▶ Reduction by equivalence

(\text{Vertex-Cover} \leq_P \text{Indept-Set} and vice versa)

▶ Reduction to a more general case

(\text{Vertex-Cover} \leq_P \text{Set-Cover})

▶ Reduction by "gadgets"

Reduction by Gadgets: Satisfiability

▶ Can we determine if a Boolean formula has a satisfying assignment?

\[(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)\]

"clause"

▶ Terminology

<table>
<thead>
<tr>
<th>Variables</th>
<th>( x_1, \ldots, x_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>( x_i ) or ( \neg x_i ) variable or its negation</td>
</tr>
<tr>
<td>Clause</td>
<td>( C = x_1 \vee x_2 \vee x_3 ) &quot;or&quot; of terms</td>
</tr>
<tr>
<td>Formula</td>
<td>( C_1 \wedge C_2 \wedge \ldots \wedge C_k ) &quot;and&quot; of clauses</td>
</tr>
<tr>
<td>Assignment</td>
<td>( (x_1, x_2, x_3) = (1, 0, 1) ) assign 0/1 to each variable</td>
</tr>
<tr>
<td>Satisfying assignment</td>
<td>( (x_1, x_2, x_3) = (1, 1, 0) ) all clauses are &quot;true&quot;</td>
</tr>
</tbody>
</table>

Reduction by Gadgets: Satisfiability

\[\text{Claim: } 3\text{-SAT} \leq_P \text{IndependentSet}.\]

\[\text{Reduction:}\]

Given 3-SAT instance \( \Phi = (C_1, \ldots, C_m) \), we will construct an independent set instance \((G, m)\) such that \( G \) has an independent set of size \( m \) iff \( \Phi \) is satisfiable

Return Yes if \( \text{solve} IS((G, m)) = \text{Yes} \)
**Reduction**

- **Idea**: construct graph $G$ where independent set will select one term per clause to be true

$$
(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \bar{x}_2 \lor x_3) \land (\bar{x}_1 \lor \bar{x}_2 \lor \bar{x}_3)
$$

- One node per term
- Edges between all terms in same clause (select at most one)
- Edges between a literal and all of its negations (consistent truth assignment)

**Correctness**

Claim: if $G$ has an independent set of size $m$, then $(C_1, \ldots, C_m)$ is satisfiable

- Suppose $S$ is an independent set of size $m$
- Assign variables so selected literals are true. Edges from terms to negations ensure non-conflicting assignment.
- Set any remaining variables arbitrarily
- At most one term per clause is selected. Since $m$ are selected, every clause is satisfied.

**Reductions So Far**

Partial map of problems we can use to solve others in polynomial time, through transitivity of reductions:

- 3-SAT
- Indep-Set
- SAT
- Vertex-Cover
- Set-Cover

**Toward a Definition of NP**

Remember our problem hierarchy:

- **Exp**
- **NP**
- **P**

What is special about the mystery problems (NP)?

**P and NP**

**Intuition**. For many “hard” decision problems, at least one thing is “easy”: if the correct answer is YES, there is an easy proof

- Independent set: show an independent set of size at least $k$
- SAT: show a satisfying assignment

**Problem classes**

- **P**: Decision problems for which there is a polynomial time algorithm.
- **NP**: Decision problems for which there is a polynomial time certifier.
  - A solution can be “certified” in polynomial time.
  - NP = “non-deterministic polynomial time”
Let $X$ be a decision problem and $s$ be problem instance (e.g., $s = (G, k)$ for Independent Set).

**Poly-time solver.** Algorithm $A(s)$ such that $A(s) = \text{Yes}$ iff correct answer is Yes, and running time polynomial time in $|s|.$

Poly-time certifier. Algorithm $C(s, t)$ such that for every instance $s,$ there is some $t$ such that $C(s, t) = \text{Yes}$ iff correct answer is Yes, and running time is polynomial in $|s|.$

- $t$ is the “certificate” or hint; size must also be polynomial in $|s|.$

**Certifier Example: Independent Set**

Input $s = (G, k).$
Problem: Does $G$ have an independent set of size at least $k$?
Idea: Certificate $t$ = an independent set of size $k$

CertifyIS($ (G, k) , t$)
if $|t| < k$ return $\text{No}$
for each edge $e = (u, v) \in E$ do
if $u \in t$ and $v \in t$ return $\text{No}$
Return $\text{Yes}$

Polynomial time? Yes, linear in $|E|.$

**Example: 3-SAT**

Input: formula $\Phi$ on $n$ variables.
Problem: Is $\Phi$ satisfiable?
Idea: Certificate $t = \text{the satisfying assignment}$

Certify3SAT($ (\Phi) , t$)
- Check if $t$ makes $\Phi$ true

**P, NP, EXP**

- 3SAT and Independent Set are in NP, as are many other problems that are hard to solve, but easy to certify!
- Claim: $P \subseteq NP$
- Claim: $NP \subseteq EXP$
- Both straightforward to prove, but not critical right now.

**NP-Complete**

- NP-complete = a problem $Y \in NP$ with the property that $X \leq_P Y$ for every problem $X \in NP!$
Cook-Levin Theorem: In 1971, Cook and Levin independently showed that particular problems were NP-Complete.
We’ll look at Circuit-SAT as canonical NP-Complete problem.

**Circuit-SAT**

Problem: Given a circuit built of AND, OR, and NOT gates with some hard-coded inputs, is there a way to set remaining inputs so the output is 1?

Satisfiable? Yes. Set inputs: 1, 1, 0.

**A Circuit-SAT reduction**

See Independent Set example in other slides

Proving New Problems NP-Complete

Fact: If Y is NP-complete and Y ≤_P X, then X is NP-complete.

Want to prove problem X is NP-complete

- Check X ∈ NP.
- Choose known NP-complete problem Y.
- Prove Y ≤_P X.
It’s easy to show that $3$-SAT $\leq_P$ Circuit-SAT. What can we conclude from this?

A. $3$-SAT is NP-complete.
B. $3$-SAT is in NP.
C. If $3$-SAT is NP-complete, then Circuit-SAT is also NP-complete.

Theorem: $3$-SAT is NP-Complete.
- In NP? Yes, check satisfying assignment in poly-time.
- Can show that Circuit-SAT $\leq_P$ 3-SAT (next time)