Polytime reductions and NP completeness

CS424, 04/07/16
We are in Chapter 7 (Time Complexity).

Recall from last time:

- $P =$ “languages decidable in polynomial time”
- $NP =$ “languages decidable in nondeterministic polynomial time”
- Example languages in NP: CLIQUE, VC, HAMPATH, SAT
- Each shown in NP using “guess and check” framework

“On input $x$:

1. *Guess* a bitstring $w$ of length $p_1(n)$ [nondeterministic]
2. *Check* $<x,w>$ have some property, in time $\leq p_2(n)$ [deterministic]”

Here $p_1(n)$ and $p_2(n)$ are polynomials in $n=|x|$, for example $p(n)=5(n+1)^3$. 
Instead of “guess and check”, Sipser uses an equivalent idea (page 293):

**Def:** A *verifier* for a language $A$ is an algorithm $V$, so that

$$A = \{ \text{ w | V accepts } <w,c> \text{ for some string } c \}$$

We say $V$ is a polynomial time verifier if it runs in time $\leq p(n)$, where $p$ is a polynomial, and $n = |w|$. We say $c$ is a *certificate* (or witness) for $w$.

Note $V$ does the “check” step of “guess and check”.

**Theorem:** Language $A$ is in NP iff it has a polynomial time verifier.

(See book.)
A function $f: \Sigma^* \rightarrow \Sigma^*$ is *polytime computable*, if there is a TM $M_f$ that on input $x$ produces output $f(x)$, and runs in time at most some polynomial $p(n)$, where $n=|x|$.

**Def:** Suppose $A$ and $B$ are languages, subsets of $\Sigma^*$. We say “$A$ is polytime reducible to $B$”, denoted “$A \leq_p B$”, if there is a polytime computable $f$ so that for all strings $x$, $x \in A$ iff $f(x) \in B$.

**Claim:** If $A \leq_p B$ and $B \in \mathsf{P}$, then $A \in \mathsf{P}$.

**Proof:** “On input $x$: compute $y=f(x)$, then run $M_B$ on $y$.” This is still polytime, since for polynomials $p_1$ and $p_2$, $p_1(p_2(n))$ is also a polynomial.
## Analogies with Chapter 5

<table>
<thead>
<tr>
<th>Intuition</th>
<th>Chapter 5</th>
<th>Chapter 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, f “easy”</td>
<td>A decidable, f computable</td>
<td>A in P, f polytime computable</td>
</tr>
<tr>
<td>A no harder than B</td>
<td>$A \leq_m B$</td>
<td>$A \leq_p B$</td>
</tr>
<tr>
<td>Harder class</td>
<td>recognizable</td>
<td>NP</td>
</tr>
<tr>
<td>A “hardest” problem</td>
<td>$A_{\text{TM}}$</td>
<td>SAT</td>
</tr>
</tbody>
</table>

**Difference:** we *know* $A_{\text{TM}}$ is not decidable, we *conjecture* SAT is not in P.
Example reduction: CLIQUE $\leq_p$ VC

Given graph $G$, its complement $G'$ is the graph with the same vertex set, but complementary edge set.

**Claim:** Suppose $G$ has $n$ vertices. Then $<G, k> \in$ CLIQUE iff $<G', n-k> \in$ VC.

**Proof:** Let $V$ be the vertex set of $G$, and $S \subseteq V$. Then $S$ is a clique in $G$ iff $V-S$ is a vertex cover in $G'$.

Function $f(<G,k>)=<G',n-k>$ is polytime computable, so CLIQUE $\leq_p$ VC. (In fact, this reduction also works the other way, from VC to CLIQUE.)
Example Reduction: $3\text{SAT} \leq_p \text{CLIQUE}$

$3\text{SAT} = \{ \langle \phi \rangle : \phi \text{ is a 3-cnf formula and } \phi \text{ is satisfiable} \}$

3-cnf means an AND of clauses, where a clause is an OR of three literals, where a literal is either a variable or negated variable.

Such a formula $\phi$ with $k$ clauses is reduced to a graph $G$ with $3k$ vertices, which has a $k$-clique iff $\phi$ is satisfiable. See book for details of this reduction!

Later we will see $\text{SAT} \leq_p 3\text{SAT}$. Putting these together, we’ll get $\text{SAT} \leq_p \text{CLIQUE}$. 
Application: SAT solvers

There are several heuristically-fast “SAT solver” software packages available. And for several hard search problems, the best known approach is to reduce the problem to an instance of SAT, and then hand that to the solver.

For examples, see work of Lu with some recent students (e.g. Siva).

NOTE: these solvers still take exponential time, in the worst case.
NP-completeness

Say language $L$ is **NP-hard** if for every $L'$ in NP, we have $L' \leq_p L$.

Say language $L$ is **NP-complete** if it is NP-hard, and in NP.

**Theorem**: Suppose $L$ is NP-complete. Then $L \in P$ iff $P=NP$.

We’ll see SAT, CLIQUE, HAMPATH, etc, are all NP-complete.

Why care about NP-completeness? Researchers have failed for decades to find efficient algorithms for NP problems, so this is strong evidence that NP-complete languages are hard.

**Or**: job security, see the Garey & Johnson cartoon (gj.pdf)
Next Time

We’ll prove the “big theorem” of Chapter 7 (Cook-Levin):

SAT is NP-complete.