CSE 347 Lecture 15

Intractability via Reductions I

1. Reductions establish NP-completeness
2. 3-SAT is NP-complete
3. Subset-Sum (& knapsack) are NP-complete

Last time we introduced NP-completeness: recall, $X$ is NP-complete if $X$ is in NP (has an efficient certifier algorithm) and every other problem in NP reduces to $X$. So, $X$ has a polynomial-time algorithm if and only if every problem in NP has a polynomial-time algorithm. As this includes many well-studied problems for which we still don’t have algorithms, this is strong (but not conclusive) evidence that the problem does not have efficient algorithms.

We saw that Circuit SAT is NP-complete, since the circuit can implement the certifier algorithms. At this point, it may seem that NP-completeness is some obscure property of problems about circuits and algorithms. The truth is quite the opposite: nearly every problem you encounter will either have an efficient algorithm, or will be NP-complete (or at least, “NP-hard” — every problem in NP reduces to it, but the problem may not be in NP.) So, NP-completeness gives a highly effective means for determining when to stop looking
for an algorithm. Moreover, once we have identified one NP-complete problem, the rest become easy to establish by reductions:

**Lemma:** Suppose \( Y \) is an NP-complete problem and \( X \) is a problem in NP. Then if \( Y \leq_p X \), \( X \) is also NP-complete.

\[
\text{Proof?} \quad \text{We only need to show that every } Z \text{ in NP reduces to } X. \quad \text{Since we know } Z \leq_p Y, \text{ there is a polynomial-time algorithm (running in } p(n) \text{ steps for a polynomial } p \text{ on inputs of size } n) \text{ for } Z \text{ using calls to } Y. \text{ We also are given that there is an algorithm for } Y, \text{ running in time } g(n) \text{ (for another polynomial } g) \text{ using calls to } X. \text{ We replace the calls to } Y \text{ with this algorithm, which is thus an algorithm for } Z \text{ using calls to } X; \text{ since each instance of } Y \text{ had size at most } p(n) \text{ (since we could not write down a larger instance in } p(n) \text{ steps) the algorithm for } Y \text{ using } X \text{ takes at most } g(p(n)) \text{ steps (in terms of the size of the original input, of } Z). \text{ Each of our steps of the original algorithm for } Z \text{ now take at most } g(p(n)) \text{ steps for an overall running time of } p(n) \cdot g(p(n)), \text{ which is another polynomial. Thus we have } Z \leq_p X \text{ for every } Z \text{ in NP.} \]

**3-SAT is NP-complete**

Now, we'll illustrate the method:

**Theorem:** 3-SAT is NP-complete.
Before we prove this, observe that since last time we proved $3\text{-SAT} \leq_p \text{Independent Set}$, we find:

**Corollary** Independent Set is NP-complete  
(By the above lemma) In turn, since we also showed Independent Set $\leq_p \text{Vertex Cover}$, we also find:

**Corollary** Vertex Cover is NP-complete

That is, the existence of a polynomial-time algorithm for any of these is equivalent to the $P$ vs. $NP$ problem, so we don’t expect that algorithms for any of these problems exist, but proving it is beyond anyone’s ability at present. So, let’s see the proof.

**Proof ($3\text{-SAT}$ is NP-complete):** We argued last time that $3\text{-SAT}$ is in $NP$. Now, we’ll show that Circuit-$\text{SAT} \leq_p 3\text{-SAT}$, which is enough by our lemma.

The main idea is to express the circuit in terms of what it says about the values carried by its wires: a fan-in 2 gate only refers to three wires. Precisely, we will have a Boolean variable for each wire of the circuit. Now, suppose an OR gate takes inputs from wires $w_i$ and $w_j$, and produces an output on wire $w_k$. The OR output puts $w_k = 1$ if $w_i$ or $w_j$ is 1, and otherwise — if both $w_i$ and $w_j$ are 0 — $w_k$ should be 0. So, we include clauses $(\neg w_i \lor w_k)$ ($\equiv \neg w_i \Rightarrow w_k$), $(\neg w_j \lor w_k)$, and $(w_i \lor w_j \lor \neg w_k)$ ($\equiv \neg (\neg w_i \land \neg w_j) \Rightarrow \neg w_k$). What about the AND and NOT gates?
For the AND, we add $(\neg w_i \lor \neg w_j)$, $(\neg w_j \lor \neg w_k)$, and $(\neg w_i \lor \neg w_k)$. Finally, if the output wire is $y$, we include the clause $(y)$ (forcing $y=1$). Observe that if the circuit was satisfiable, then for the settings of the wires we obtain from evaluating the circuit, each of the clauses we added for each gate is satisfied. Conversely, if the 3-CNF we produced is satisfied, then (properly, by induction on the number of layers of gates in the circuit) we observe that since the inputs $w_i$ and $w_j$ are forced to take the same values as they would take in the evaluation of the circuit (on input $x_1, \ldots, x_n$) and since no matter what values $w_i$ and $w_j$ take, one of the clauses we added is only satisfied by a unique value of $w_k$ that matches the value $w_k$ would take in the circuit if values $w_i$ and $w_j$ were given as input, $w_k$ is also forced to take the same value as in the evaluation of the circuit. Thus, the final clause $(y)$ is satisfied if and only if the circuit would output 1 on the given setting of the inputs $x_1, \ldots, x_n$. Observe that given the formula, we can generate this circuit very easily—we merely need to generate a fixed set of clauses for each gate type, with the correct wire numbers plugged in. So this can be passed to our 3SAT call in polynomial time, and we argued above that the answers are the same. Thus, Circuit SAT $\leq_p$ 3SAT, so 3SAT is NP-complete. ☑
Notice, something amazing has happened—instead of needing to find a "weakness" of every algorithm for 3-SAT, Independent Set, etc., we merely need to show how to use such an algorithm to design another algorithm (that seems too powerful to exist).

**Subset-Sum (or Knapsack) are NP-complete.**

Proving NP-completeness is generally easier when the NP-complete problem is more similar to the problem of interest. For example, here is a problem about numbers: Subset Sum is essentially the special case of Knapsack where weights and values are the same: given a set of numbers \( w_1, w_2, \ldots, w_n \), is there a subset \( S \) that sums to \( W \)?

**Theorem** Subset-Sum is NP-complete.

Note: this shows that the exponential dependence on the size of the numbers in our dynamic programming algorithm for Knapsack is probably unavoidable.

**Proof:**

1. **Subset-Sum is in NP because...** we could use a (sorted) list of indices \( S \subseteq \{1, \ldots, n\} \) such that \( \sum_{i \in S} w_i = W \) as a certificate. We can check that the indicated subset sums to \( W \) in polynomial time, and indeed the very definition of the problem is to decide whether or not there is such a subset.

2. **We'll show 3SAT \( \leq_p \) Subset-Sum, and use our lemma.** Let \( C_1 \land C_2 \land \cdots \land C_m \) be our 3-CNF on \( x_1, \ldots, x_n \). We'll use \( n+m \)-digit numbers in which the number \( w_{i,0} \) captures the effect of setting \( x_i \) to \( 0 \), and \( w_{i,1} \) captures...
setting $x_i$ to 1—in the first $n$ digits we track which of $x_1, \ldots, x_n$ are set to a value, and in the final $m$ digits, we track which of $C_1, \ldots, C_m$ are satisfied: suppose $x_i$ appears in $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$ and $7x_i$ appears in $C_{j_1}, C_{j_2}, \ldots, C_{j_k}$.

Then our numbers for $x_i$ are
\[
\begin{align*}
& w_{i,1}: 00 \ 0 \ 0 \ 0 \ 0 \\
& w_{i,0}: 00 \ 1 \ 0 \ 0 \ 0
\end{align*}
\]

If we assign $x_1=b_1, x_2=b_2, \ldots, x_n=b_n$, $\sum w_{i,b_i}$ has a 1 in each of the first $n$ digits and the number of satisfied literals in clause $j$ in digit $n+j$. So that we can hit a fixed target, we will include an additional $|C_j|-1$ numbers of the form $\underbrace{0 \ 0 \ 0 \ \cdots \ 0}_{n+m}$ in the set. Now, our target $W = \underbrace{11 \ 1 \ | \ c_1, c_2, \ldots, c_m \ 1}$; observe that for any satisfying assignment, $x_1=b_1, x_n=b_n$, we can add, for each $j$, $|C_j|-(\text{# sat literals in } C_j)$ of the “padding numbers” for $C_j$ to $\sum w_{i,b_i}$ to obtain precisely the target $W$. Conversely, we first observe that (a) we have precisely two numbers with a 1 in the 1st $n$ digits (the rest are 0) and (b) we have only $|C_j|+|C_j|-1$ numbers that are 1 in any of the last $m$ digits, where $|C_j| \leq 3$, so these digits sum to at most 5. Thus, there are no carries in the sum. So, we can only hit $W$ by choosing exactly one of $w_{i,0}$ or $w_{i,1}$ for each $i$. Moreover, since we have at most $|C_j| - 1$ padding numbers for each $C_j$, if we obtain $|C_j|$ in digit $n+j$, there must have been some $w_{i,b_i}$ with a 1 in position $n+j$—that is, $x_i=b_i$ satisfies $C_j$, if
by construction. Since $C_1, \ldots, C_m$ are therefore satisfied by $x_1 = b_1, \ldots, x_n = b_n$, $C_1 \land \cdots \land C_m$ is satisfiable, as needed.

Finally, we observe that we can construct these numbers easily in time $O(n(n+m))$, so indeed, $\text{3SAT} \leq_p \text{Subset-Sum}$.\"