NP-Completeness

Lecture by Russell Impagliazzo
Notes by William Matthews

Lecture April 26, 2010

Last time, we saw a “meta-computing” $NP$-complete problem, $CIRCUIT$–$SAT$. Today, we will look
at reductions between $NP$ problems with the goal of showing first that there are combinatorial $NP$-complete
problems. After that, we’ll show that not only do such problems exist, but they’re fairly pervasive.

Let $L_1$ be a known $NP$-complete language, and let $L_2$ be a language which we wish to prove $NP$-complete.

\[
L_1 = \{ x | \exists y \ C_1(x,y) \} \\
L_2 = \{ x | \exists y \ C_2(x,y) \}
\]

We wish to find a reduction $f$ which runs in polynomial time such that for all $x$

\[
x \in L_1 \iff f(x) \in L_2
\]
equivalently

\[
\exists y \ C_1(x,y) \iff \exists y' \ C_2(f(x),y')
\]

To prove the correctness of the reduction $f$, we often find it useful to show the existence of the following two
maps between the “solutions” to the two corresponding search problems

\[
g : x, y_1 \rightarrow y_2 \\
h : x, y_2 \rightarrow y_1
\]
such that for all $x$

\[
C_1(x, y_1) \implies C_2(f(x), g(y_1)) \\
C_2(f(x), y_2) \implies C_1(x, h(y_2))
\]

The map $g$ allows us to show that given a solution to $x \in L_1$, we get a solution to the corresponding
$f(x) \in L_2$. The map $h$ allows to show that given a solution to $f(x) \in L_2$, we get a solution to $x \in L_1$.
Note that these maps do not need to be computable in polynomial time since they’re just used to prove
correctness.

Recall $CIRCUIT$–$SAT$. An instance is a circuit $C(x_1, x_2, \ldots, x_n)$ on inputs $x_1, \ldots, x_n$. A solution is
of the format $x_1 = v_1, \ldots, x_n = v_n$, and the constraint is that $C(v_1, \ldots, v_n) = 1$.

Define a literal as a variable or its negation: $x_i, \neg x_i \equiv \overline{x_i}$. A clause is an “or” of literals: $x_1 \lor x_2 \lor x_3$. A $k$-clause is an “or” of at most $k$ literals (the previous example was a 3-clause). A CNF is an “and” of clauses: $c_1 \land c_2 \land \cdots \land c_m$. A $k$-CNF is a CNF of $k$-clauses.

Define the problem $k$-SAT. An instance is a circuit $C(x_1, x_2, \ldots, x_n)$ where $C$ is a $k$-CNF on inputs
$x_1, \ldots, x_n$. A solution is of the format $x_1 = v_1, \ldots, x_n = v_n$, and the constraint is that $C(v_1, \ldots, v_n) = 1$. Equivalently, a solution satisfies the constraints that for all clauses $c_i = \ell_1 \land \ell_2 \land \ell_3$, at least one of the literals $\ell_i = 1$. 

1
**Theorem 1** (Cook-Levin Theorem). 3-SAT is NP-complete.

**Proof.** Need to give $f : C(x_1, \ldots, x_n) \rightarrow \phi(x_1, \ldots, x_n, y_{n+1}, \ldots, y_m)$ where $\phi$ is a 3-CNF. Recall the format of $C$: The first $n$ gates are the inputs $g_1 = x_1, \ldots, g_n = x_n$, and then each of the remaining gates $g_{n+1}, \ldots, g_m$ is of the form $g_k = op_k(g_j, g_l)$ where $op_k$ is either $\vee$ or $\wedge$ and $j, k < i$. We will use the variables $y_{n+1}, \ldots, y_m$ to encode the statements “$y_k = op_k(y_j, y_l)$.” How can we encode these statements as a 3-CNF? Each statement $y_k = op_k(y_j, y_l)$ means “if $y_j = b_1$ and $y_l = b_2$ then $y_k = op_k(b_1, b_2)$” which is equivalent to “$y_j = 0 \Rightarrow y_k = 1$” or “$y_l = 0 \Rightarrow y_k = 0$.” By enumerating the 4 possible values for $b_1, b_2$ we get 4 clauses. For example if $g_k = g_j \vee g_l$ then we get the following 4 clauses:

- $(b_1 = 0, b_2 = 0) : y_j \wedge y_l \wedge \overline{y_k}$
- $(b_1 = 0, b_2 = 1) : y_j \wedge \overline{y_l} \wedge \overline{y_k}$
- $(b_1 = 1, b_2 = 0) : \overline{y_j} \wedge y_l \wedge \overline{y_k}$
- $(b_1 = 1, b_2 = 1) : \overline{y_j} \wedge \overline{y_l} \wedge y_k$

We construct the similar set of 4 clauses when $g_k = g_j \wedge g_l$. Finally, we add a clause $g_m$ to ensure that the circuit is satisfiable. Since we do a constant amount of work for each gate of $C$, the reduction runs in polynomial time.

Next, we must show that if there exists a solution to the CIRCUIT-SAT problem $C$, then there exists a solution to the 3-SAT problem $\phi$, and show that if there exists a solution to the 3-SAT problem $\phi$, then there exists a solution to the CIRCUIT-SAT problem $C$.

(First direction.) Assume $\exists x_1, \ldots, x_n$ such that $C(x_1, \ldots, x_n) = 1$. We must show how we can get values that satisfy $\phi$. Let $x_1, \ldots, x_n$ be values for $x_1, \ldots, x_n$ in $\phi$, and let $w_i$, for $n + 1 \leq i \leq n$, be the value for gate $g_i$ in $C(x_1, \ldots, x_n)$. Equivalently let $w_i = \overline{g_i}(w_j, w_k)$. Setting $y_{n+1} = w_{n+1}, \ldots, y_m = w_m$ satisfies all of the clauses corresponding to gates. Since $C(x_1, \ldots, x_n) = 1$, we know $w_n = 1$ so the last clause is satisfiable.

(Second direction (sketch).) Assume $\exists x_1, \ldots, x_n, y_{n+1}, \ldots, y_m$ such that $\phi(x_1, \ldots, x_n, y_{n+1}, \ldots, y_m)$ is true. In particular, this means that for each $i > n$ we know that the clauses corresponding to each gate $g_i$ are satisfied. Thus, $w_i$ must be the value of gate $g_i$ in $C(x_1, \ldots, x_n)$ (by induction on $i$). Finally, since we know that $w_{n+1}$ must be true to satisfy the last clause that the output of $C(x_1, \ldots, x_n)$ must be true. Thus $x_1, \ldots, x_n$ must be a solution to $C$.

Thus, 3-SAT is NP-hard. Since 3-SAT is a special case of CIRCUIT-SAT (or simply by looking at the definition of 3-SAT), we know 3-SAT $\in$ NP.

**Big-Independent-Set Problem:** Instance: graph $G = (V, E)$ and $1 \leq k \leq n = |V|$. Solution: $S \subseteq V, |S| = k$. Constraint: $S$ is an independent set in $G$, i.e., $\forall e \in E$ either (or both) $u \notin S$ or $v \notin S$.

We wish to show that Big-Independent-Set is NP-hard.

**Proof.** We begin by constructing a map from 3-CNF formulas to graphs and $k$.

Consider $\phi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \cdots \wedge (\ell_{n,1} \vee \ell_{n,2} \vee \ell_{n,3})$, where each $\ell_{i,j}$ is either $x_p$ or $\overline{x_p}$ for $1 \leq p \leq n$.

Think of a solution as picking one literal per clause, subject to the constraint that we never pick both $x_p$ and $\overline{x_p}$.

Let $V = \{(i, \ell) \mid \ell \text{ appears in clause } i\}$, and $E = \{((i, \ell), (i, \ell')) \mid \ell \neq \ell'\} \cup \{((i, \ell), (i', \ell)) \mid i \neq i'\}$. Choose $k = m$ (we want to choose 1 literal per clause, subject to no contradictions).

Assume $\phi(x_1, \ldots, x_n) = 1$. Then we know that each clause has at least one true literal. For each $i$, pick $(i, \ell)$ where $\ell$ is the first true literal in the $i^{th}$ clause (for some ordering). Since we choose only one vertex
per clause, we don’t violate any edges of the first type. Since we can’t have both \( \ell \) and \( \bar{\ell} \) true, we know we don’t violate any clauses of the second type. Since we choose exactly \( m \) vertices, we get an independent set of exactly that size.

Assume \( S \) is an independent set in \( G \) of size \( m \). Since \( |S| = m \) and no two endpoints with the same value of \( i \) can be in \( S \) (by edges of type 1), we know that \( S \) has exactly one vertex per clause in \( \phi \). Let \( v_p = 1 \) if \( \exists (i, x_p) \in S \), and \( v_p = 0 \) if \( \exists (i, \bar{x}_p) \in S \). ( Arbitrarily) set \( v_p = 0 \) otherwise. Since we can’t have both \( (i, \ell) \) and \( (\bar{i}, \bar{\ell}) \) in \( S \), \( v_p \) is well defined. Since each clause \( i \) has some \( (i, \ell) \in S \), there must be some \( x_p \) which is set so that clause \( i \) is satisfied.

\[ \square \]