NP-Completeness

Lecture by Russell Impagliazzo
Notes by William Matthews

Lecture April 26, 2010

Last time, we 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.

$$L_1 = \{x \mid \exists y\ C_1(x, y)\}$$
$$L_2 = \{x \mid \exists y\ C_2(x, y)\}$$

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

$$g : x, y_1 \to y_2$$
$$h : x, y_2 \to y_1$$

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 $C(x_1, x_2, \dots, x_n)$ on inputs $x_1, \dots, x_n$. A solution is of the format $x_1 = v_1, \dots, x_n = v_n$, and the constraint is that $C(v_1, \dots, v_n) = 1$.

Define a *literal* as a variable or its negation: $x_i, \neg x_i \equiv \overline{x_1}$. A *clause* is an "or" of literals: $x_1 \vee \overline{x_2} \vee x_3$. 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: $c_1 \wedge c_2 \wedge \cdots \wedge c_m$. A k-CNF is a CNF of k-clauses.

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

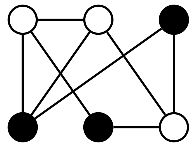

Figure 1: The black nodes are an example of an independent set of size 3.

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 ϕ 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 $g_i = op_i(g_j, g_k)$ where op_i is either $\vee$ or $\wedge$ and $j, k < i$. We will use the variables $y_{n+1}, \ldots, y_m$ to encode the statements "$y_i = op_i(y_j, y_k)$." How can we encode these statements as a 3-CNF? Each statement $y_i = op_i(y_j, y_k)$ means "if $y_j = b_1$ and $y_k = b_2$ then $y_i = op_i(b_1, b_2)$" which is equivalent to "$y_j = \neg b_1$ or $y_k = \neg b_2$ or $y_i = op_i(b_1, b_2)$." By enumerating the 4 possible values for b_1, b_2 we get 4 clauses. For example if $g_i = g_j \wedge g_k$ then we get the following 4 clauses:

$$\begin{array}{ll}
(b_1 = 0, b_2 = 0) & y_j \wedge y_k \wedge \overline{y_i} \\
(b_1 = 0, b_2 = 1) & y_j \wedge \overline{y_k} \wedge \overline{y_i} \\
(b_1 = 1, b_2 = 0) & \overline{y_j} \wedge y_k \wedge \overline{y_i} \\
(b_1 = 1, b_2 = 1) & \overline{y_j} \wedge \overline{y_k} \wedge y_i
\end{array}$$

We construct the similar set of 4 clauses when $g_i = g_j \vee g_k$. 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 ϕ, and show that if there exists a solution to the 3-SAT problem ϕ, then there exists a solution to the $CIRCUIT - SAT$ problem C.

(First direction.) Assume $\exists v_1, \ldots, v_n$ such that $C(v_1, \ldots, v_n) = 1$. We must show how we can get values that satisfy ϕ. Let $v_1, \ldots, v_n$ be values for $x_1, \ldots, x_n$ in ϕ, and let w_i, for $n+1 \leq i \leq m$, be the value for gate g_i in $C(v_1, \ldots, v_n)$. Equivalently let $w_i = op_i(w_j, w_k)$. Setting $y_{n+1} = w_{n+1}, \ldots y_m = w_m$ satisfies all of the clauses corresponding to gates. Since $C(v_1, \ldots, v_n) = 1$, we know $w_m = 1$ so the last clause is satisfiable.

(Second direction (sketch.)) Assume $\exists v_1, \ldots, v_n, w_{n+1}, \ldots, w_m$ such that $\phi(v_1, \ldots, v_n, w_{n+1}, \ldots, w_m)$ is true. In particular, this means that for each $i > n$ we now that the clauses corresponding to each gate g_i are satisfied. Thus, w_i must be the value of gate g_i in $C(v_1, \ldots, v_n)$ (by induction on i.) Finally, since we know that w_{n+1} must be true to satisfy the last clause that the output of $C(v_1, \ldots, v_n)$ must be true. Thus $v_1, \ldots, v_n$ 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. $\forall e = \{u, v\} \in E$ 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 $\phi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \cdots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m3})$, 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 $E = \{\{(i, \ell_1), (i, \ell_2)\} \mid \ell_1 \neq \ell_2\} \cup \{\{(i, \ell), (i', \overline{\ell})\} \mid i \neq i'\}$. Choose $k = m$ (we want to choose 1 literal per clause, subject to no contradictions).

Assume $\phi(v_1, \ldots, v_n) = 1$. Then we know that each clause has at least one true literal. For each i, pick (i, ℓ) where ℓ is the first true literal in the i^{th} clause (for some ordering). Since we choose only one vertex

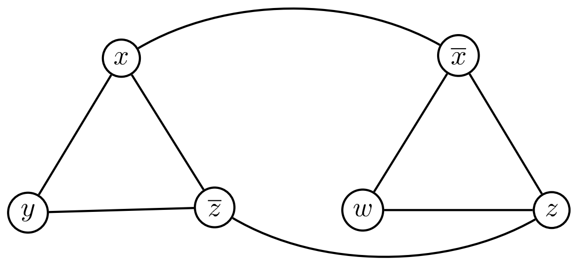

Figure 2: Graph corresponding to $(x \vee y \vee \overline{z}) \wedge (\overline{x} \vee w \vee z)$: A triangle for each clause, and an edge between each pair of contradictory literals in different clauses.

per clause, we don't violate any edges of the first type. Since we can't have both ℓ and $\overline{\ell}$ 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 ϕ. Let $v_p = 1$ if $\exists(i, x_p) \in S$, and $v_p = 0$ if $\exists(i, \overline{x_p}) \in S$. (Arbitrarily) set $v_p = 0$ otherwise. Since we can't have both (i, ℓ) and $(i', \overline{\ell})$ 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.

$\square$