ICS 260—Fall, 2000–Class Notes 12: Reductions and Completeness

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

1 Reductions

Given that $P \subseteq NP$, it is now a natural question to ask if P in fact differs from NP. Are there problems in NP which are not in P? That is, are there intractable problems in NP?

Until it is proven than $P \neq NP$, there is no hope of proving that a particular problem L is in $NP - P$. Thus, we are just going to prove results of the following form: if $P \neq NP$, then $L \in NP - P$. This at least gives strong evidence that the problem L is intracable, because if L is not intractable, then there is an efficient algorithm for *every* problem in NP.

The main tool which we will need for this type of result is a *polynomial transformation* (also called a *polynomial reduction*). A polynomial transformation from problem $L_1 \subseteq \Sigma_1^*$ to problem $L_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following conditions:

1. There is a deterministic polynomial time TM which computes f.
2. For all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

If there is a polynomial transformation from $L_1$to L_2, then this is denoted by $L_1 \propto L_2$.

The importance of this idea follows from the following lemma:

Lemma 1 *If $L_1 \propto L_2$, then $L_2 \in P$ implies that $L_1 \in P$ (and equivalently $L_1 \notin P$ implies that $L_2 \notin P$).*

Proof. Let Σ_1 and Σ_2 be the alphabets for L_1 and L_2 respectively. Since $L_1 \propto L_2$, there is a polynomial time TM M_f which computes a function $f : \Sigma_1^* \to \Sigma_2^*$ such that $x \in L_1$ if and only if $f(x) \in L_2$. Furthermore, M_f runs in time bounded by a polynomial p_f. We also have a TM M_2 which decides L_2 in time which is bounded by a polynomial p_2. We can construct a TM M_1 which computes L_1 which is a composition of M_f and M_2. M_1 works as follows: on input x, it first runs M_f and computes $f(x)$. Then it runs M_2 and decides whether $f(x) \in L_2$. Since $f(x) \in L_2$ if and only if $x \in L_1$, M_1 computes L_1. The TM also runs in polynomial time. The first part takes time $p_f(|x|)$. The output of M_f has length at most $p_f(|x|)$, so then the second part takes time $p_2(p_f(|x|))$. Thus the new TM takes time at most $p_f(|x|) + p_2(p_f(|x|))$ which is a polynomial in $|x|$. ■

We will now give an example of a very simple polynomial transformation. First, we need to define an additional problem. Recall that if we have an undirected graph G, then a *simple* cycle in G is a sequence of distinct vertices $\langle v_1, v_2, \ldots, v_k \rangle$ such that for $1 \leq i < k$, $\{v_i, v_{i+1}\} \in E$ and $\{v_k, v_1\} \in E$. A *Hamiltonian cycle* is a simple cycle that includes all the vertices of G. The Hamiltonian cycle problem is defined as follows:

Input: A undirected graph $G = (V, E)$.

Question: Does G have a Hamiltonian cycle?

Proposition 2 $HC \propto TSP$

Proof. In order to show that $HC \propto TSP$, we need to show a function f which is polynomial time computable which takes as input an undirected graph G and produces an instance of TSP (i.e. a set of cities, distances between the cities and a bound B). Furthermore, it must be the case that G has a Hamiltonian cycle if and only if there is a tour of the cities of length at most B.

In this case, the function f is quite simple. There is a city c_i for each node v_i in the graph. For every city c_i, $d(c_i, c_i) = 0$. For $i \neq j$, the distance between city i and city j $d(c_i, c_j)$ is 1 if $\{v_i, v_j\} \in E$ and is 2 otherwise. Finally, we must specify the bound B which is just n, the number of nodes in the graph G.

It is straight-forward to see that the transformation can be computed in polynomial time. For each of the $n(n-1)/2$ distances which must be computed, we need only check whether the pair of cities corresponds to an edge in G. To get the bound B, we need only count the vertices in G. Thus, in order to prove that this is in fact a polynomial transformation, we must now prove that G has a Hamiltonian cycle if and only if the set of cities has a tour with total distance at most n.

Let's say that G does have a Hamiltonian cycle $\langle v_{i_1}, v_{i_2}, \ldots, v_{i_n} \rangle$. Then we claim that $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_n} \rangle$ is a tour with length at most n. The tour is a valid tour since the cycle is a simple cycle and no vertex is repeated. Furthermore, the tour hits all n cities since the cycle is Hamiltonian and includes every vertex in the graph. The tour then traverses n inter-city distances. Each of these distances is 1, since each city pair in the tour corresponds to an edge in the graph. That is, since $\{v_{i_j}, v_{i_{j+1}}\}$ is an edge, then $d(c_{i_j}, c_{i_{j+1}}) = 1$.

Similarly, if there is a tour of length at most n of the set of cities, it must correspond to a Hamiltonian cycle. It must be the case that each inter-city distance in the tour has distance 1. There are n cities, and every inter-city distance is either 1 or 2. If even a single inter-city distance in the tour is 2, then the tour must have length greater than n. ∎

The essence of what we have shown here is that TSP is at least as hard as HC. That is to say, using Lemma 1 we know that if there is a polynomial time algorithm for TSP, then there is a polynomial time algorithm for HC.

In the case of this particular reduction, the two problems closely resemble one another and this points the way to the right reduction. In examples we will see later, we will prove resuctions between problems that seem unrelated. Often these proofs are much more involved than the proof we saw above.

The polynomial transformability relation is transitive as stated in the following lemma:

Lemma 3 *If $L_1 \propto L_2$ and $L_2 \propto L_3$, then $L_1 \propto L_3$.*

Proof. Let Σ_1, Σ_2 and Σ_3 be the alphabets for L_1, L_2 and L_3, respectively. We have a polynomially computable function $f_1 : \Sigma_1^* \to \Sigma_2^*$ such that $x \in L_1$ if and only if $f_1(x) \in L_2$. We have a polynomially computable function $f_2 : \Sigma_2^* \to \Sigma_3^*$ such that $x \in L_2$ if and only if $f_2(x) \in L_3$. We can compose the two functions to get $f_2(f_1(x))$ such that $x \in L_1$ if and only if $f_2(f_1(x)) \in L_3$. The proof that $f_2(f_1)$ can be computed in polynomial time is analogous to the proof of Lemma 1. ∎

Two problems L_1 and L_2 are said to be *polynomially equivalent* if $L_1 \propto L_2$ and $L_2 \propto L_1$. Polynomial equivalence is an equivalence relation. Furthermore, the $\propto$ relation yields a partial ordering over the equivalence classes. In this case, the class P forms the least equivalence class in that it contains the easiest problems in the class. We shall now see that the NP-complete problems form the hardest problems in the class.

A language L is said to be *NP-complete* if $L \in NP$ and for every $L' \in NP$, $L' \propto L$. In other words, L is at least as hard as every problem in NP. If any NP-complete problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time. Yet another way to say this is that if $P \neq NP$, then $L \in NP - P$. This is equivalent to saying that if $L \in P$, then $P = NP$.

At the outset, it looks like it is very difficult to prove that a problem is NP-complete, because one must show a reduction for *every* problem in NP. Actually, once we have our first NP-complete problem, proving another problem is NP-complete becomes a much easier task. This results from the following lemma:

Lemma 4 *Let L_1 and L_2 be two problems in NP. If L_1 is NP-complete and $L_1 \propto L_2$, then L_2 is also NP-complete.*

Proof. We have to prove that for every $L' \in NP$, $L' \propto L_2$. Since L_1 is NP-complete, we know that $L' \propto L_1$. Using the transitivity of $\propto$ and the fact that $L_1 \propto L_2$, we get that $L' \propto L_2$. ∎

This gives us much more straightforward approach to proving that a problem is NP-complete. To show that a language L is NP-complete, we must show two things:

1. $L \in NP$
2. For some other NP-complete problem L', $L' \propto L$.

2 SAT and Cook's Theorem

The problem of boolean satisfiability plays a special role in the theory of NP-completeness since it was the first problem to be shown NP-complete. We will call the problem SAT for short.

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of boolean variables. A truth assignment for U is a function $t : U \to \{T, F\}$. u_i is *true* if and only if $t(u_i) = T$ and u_i is *false* if and only if $t(u_i) = F$. The variable u_i gives rise to two *literals*, u_i and $\bar{u}_i$. The literal u_i is true if and only if the variable u_i is true. The literal $\bar{u}_i$ is true if and only if the variable u_i is false.

A *clause* over U is a set of literals over U. The clause represents a disjunction over the literals in the set, so the clause is satisfied under a truth assignment if and only if at least one of the literals is true. A collection C of clauses over U is satisfiable if and only if there is a single truth assignment t which simultaneously satisfies all of the clauses.

Here is the problem SAT

Input: A set of variables U and a set C of clauses over the variables.

Question: Is C satisfiable?

Cook's theorem established the first NP-complete problem.

Theorem 1 (Cook's Theorem) *SAT is NP-complete.*