CS 482 Summer 2004

Proving that Problems are NP-Complete

To prove that a problem X is NP-Complete, you need to show that it is both in NP and that it is NP-Hard. Steps 2 through 5 seek to accomplish the latter.

Step 1: Show that X is in NP. We want to argue that there is a polytime verifier for X. In other words, for any yes instance of X, there exists a certificate that the verifier will accept, and for any no instance of X, there is no certificate that the verifier will accept. Both the size of the certificate and the running time of the verifier must be polynomial. Often this step is very brief, but necessary.

Step 2: Pick a known NP-Complete problem. State what problem Y you are reducing to X. You need to show that $Y \leq_p X$. You may use any problem Y which we have proved in class to be NP-Complete, as well as any problem you have proved to be NP-Complete on the homework assignments. Some problems will be far easier to use than others in your proof.

Step 3: Construct an algorithm to solve Y given an algorithm to solve X. You need to show that any instance of Y can be solved using a polynomial number of operations, and a polynomial number of calls to a black box that can solve X. It is very easy to get mixed up and instead prove that $X \leq_p Y$. Unfortunately, this is not what you want to show (we already know that Y is NP-Complete).

Step 4: Prove the correctness of your algorithm. This has 2 parts: You want to show that given a yes instance of Y your algorithm returns "yes". Furthermore, you want to show that given a no instance of Y, your algorithm returns "no". It is always trivial to come up with an algorithm that satisfies just one of these two conditions. We want something that satisfies both.

Step 5: Polytime and wrap-up. Finally, you need to conclude that since your algorithm runs in polynomial time, $Y \leq_p X$. Since Y is NP-Complete, X is NP-Hard, and since we also have shown that X is in NP, X is in fact NP-Complete.

An Example: INDEPENDENT SET (IS) is NP-Complete

First, IS is in NP, since given any set S we can check in polytime that S is independent and that $|S| = k$. So for a yes instance, we simply use an independent set of size k.

And for a no instance, it is clear that no such set exists. Therefore IS is in NP. Now we will show that IS is NP-Hard via reduction to 3-SAT.

Suppose we have an instance $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where C_i is the disjunction of 3 variables, drawn from $x_1, x_2, \ldots, x_n$ and their negations, $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$. We create the graph G as follows:

For each variable in each clause, create a node, which we will label with the name of the variable. Therefore there may be multiple nodes with the label x_i or $\overline{x_i}$, if these variables appear in multiple clauses.For each clause, add an edge between the three nodes corresponding the variables from that clause. We will call these three nodes and three edges a "clause gadget". Finally, for all i, add an edge between every pair of nodes with one labeled x_i and the other labeled $\overline{x_i}$. Now use our black box for IS to determine whether or not there is an independent set of size m in G. If there is, return yes. Otherwise, return no.

We must now show that this algorithm correctly determines whether or not F has a satisfying assignment. Suppose F has a satisfying assignment A. Then at least one variable in each clause is satisfied by A. Define S to be a set of nodes in G found by selecting one of the satisfied variable nodes in each clause gadget. Since we picked one node for each clause, there are clearly m nodes in S. Furthermore, since we only select a single node for each gadget, independence is not violated within any clause gadget. Finally, independence is not violated between clauses, since the only edges between clauses go between nodes with labels x_i and $\overline{x_i}$, and A can not have satisfied both of these, so we never would have selected both. Therefore S is an independent set of size k, so our algorithm will return yes.

Now suppose that our algorithm returns yes. We now need to show that there is a satisfying assignment A. Since our algorithm returned yes, there must be an independent set S of size m on G. Since S is independent, at most one node in each clause gadget must be used by S. But in fact, since there are exactly m clause gadgets, S must contain exactly one node from each clause gadget. Since S is independent, no pair of nodes x_i and $\overline{x_i}$ are ever both selected for S. Consider the following assignment. Set $A(x_i) = T$ if $x_i \in S$ and set $A(x_i) = F$ otherwise. Observe that this is a truth assignment, since all variables are assigned either T or F, but not both. Furthermore, A satisfies F, since by our construction, A satisfies each clause.

Therefore we have shown that 3-SAT $\leq_p$ IS. Thus IS is NP-Hard, and since we have shown IS to be in NP, IS is NP-Complete.