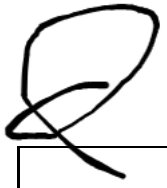

Introduction to Algorithms **Proving NP-Completeness**
CS 482 Spring 2006

To prove a problem X is NP-complete, you need to show that it is both in NP and that it is at least as hard any other problem in NP. This last step is typically done by showing that $Y \leq_P X$ for some problem Y that you already know to be NP-Complete. This is described in steps 2 through 5.

Step 1: Show that X is in NP. We want to argue that there is an efficient certifier for X. In other words, there is a certifier such that for any yes instance of X, there exists a certificate that the certifier will accept, and for any no instance of X, there is no certificate that the certifier will accept. The running time of certifier (and hence the size of the certificate) must be polynomial. Typically, a solution to the given problem is a sufficient certificate. 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. By the very definition of NP-complete, if X is NP-complete, then you can technically use any other NP-complete problem Y to show this. However, some problems will be far easier to use than others in your proof. It is often useful to spend some time thinking about which of the problems you know to be NP-Complete would be most natural to use for a give reduction.

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. **Note:** 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).

Typically, you will show how to solve Y by constructing a single input to the black box for X, and typically, your algorithm will output the same answer as is given by the black box. However, this is not always the case.

Step 4: Prove the correctness of your algorithm. This requires proving an if and only if statement. You want to show that given a yes instance of X your algorithm returns "yes", *and* you want to show that if your algorithm returns "yes," then the given input is indeed a yes instance of X. 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 (typically a construction to be used with the black box for Y) runs in polynomial time, $Y \leq_p X$. Since Y is NP-complete, and since we also have shown that X is in NP, X is NP-complete.

An Example: Set Cover

Problem: An instance of Set Cover is given by a ground set $U = x_1, x_2, \ldots, x_n$, a collection of m subsets $S_i \subseteq U$ of that ground set, and an integer k. The question is, can you select a collection C of at most k of these subsets such that taken together, they "cover" all of U? In other words, is there a

set $C \subseteq \{1, 2, \ldots, m\}$ such that $|C| \leq k$ and

$$\bigcup_{i \in C} S_i = U.$$

Theorem: Set Cover is NP-Complete.

Proof: First, we argue that Set Cover is in NP, since given a collection of sets C, a certifier can efficiently check that C indeed contains at most k elements, and that the union of all sets listed in C does include all elements from the ground set U.

We will now show that Vertex Cover $\leq_P$ Set Cover. Given an instance of Vertex Cover (i.e. a graph $G = (V, E)$ and an integer j), we will construct an instance of the Set Cover problem. Let $U = E$. We will define n subsets of U as follows: label the vertices of G from 1 to n, and let S_i be the set of edges that incident to vertex i (the edges for which vertex i is an end-point). Note that $S_i \subseteq U$ for all i. Finally let $k = j$. This construction can be done in time that is polynomial in the size of the Vertex Cover instance. We now run our black box for the Set Cover problem and return the same result it gives.

To prove that this answer is correct, we simply need to show that the original instance of Vertex Cover is a yes instance if and only if the Set Cover instance we created is also a yes instance.

Suppose G has a vertex cover of size at most j. Let S be such a set of nodes. By our construction, S corresponds to a collection C of subsets of U. Since we defined $k = j$, C clearly has at most k subsets. Furthermore, we claim that the sets listed in C cover U. To see this, consider any element of U. Such an element is an edge e in G, and since S is a vertex cover for G, at least one of es endpoints is in S. Therefore C contains at least one of the sets associated with the endpoints of e, and by definition, these both contain e.

Now suppose there is a set cover C of size k in our constructed instance. Since each set in C is naturally associated with a vertex in G, let S be the set of these vertices. $|S| = |C|$ and thus S contains at most j nodes. Furthermore, consider any edge e. Since e is in the ground set U and C is a set cover, C must contain at least one set that includes e. But by our construction, the only sets that include e correspond to nodes that are endpoints of e. Thus, S must contain at least one of the endpoints of e.

We have now shown that our algorithm solves the Vertex Cover problem using a black box for the Set Cover problem. Since our contruction takes polynomial time, and we have shown that Set Cover is in NP, we can conclude that Set Cover is NP-Complete.

This particular proof was fairly easy, because, as the proof indicates, Vertex Cover is basically a special case of Set Cover. Note that showing that a general instance of Set Cover can be solved using a black box for Vertex Cover would be much more difficult (although clearly possible, since both problems are NP-Complete).