Class Note #21

Date: 04/03/2006

[Overall Information]
In this class, we introduced the notion of “reductions” to formalize the notion that one problem is “at least as difficult as” another. We introduced the Independent Set and Vertex Cover problems, and saw our first formal example of a reduction, from Independent Set to Clique.

[The Only Announcements Today]
The fifth quiz is scheduled on 04/12/2006. It will cover the concept of P, NP and reductions. Before the quiz, there will be one more homework set which will be assigned in the next lecture in Wednesday. The corresponding homework solution will be given next Monday.

[During the Lecture]
1. After today’s class began, we reviewed the concept of NP and proof verification as well as its importance. The first question in today’s lecture is what does it mean for problem X to be at least as difficult as problem Y.

   (1) The first guess is to use the relation of superset and subset. Since such way might make the original problem harder or easier, nothing can
be guaranteed. (For instance, the language of all strings is very easy to
decide: always answer “Yes”. The language of no strings is equally easy:
always answer “No”. So the “in between” languages can be much more
difficult.)

(2) Another way suggested was to prove that somehow solving Y is a
“byproduct” of solving X. But this notion may be difficult to do anything
with, as there may be many different ways to solve X, some including
solving Y, and others not.

2. The second attempt is heading for the correct direction. The intuition
that works out well is: if being able to solve X would let us solve Y
efficiently, then X must be at least as difficult as Y. Of course, we still
need to formalize the notion of “being able to solve X would let us solve
Y”.

what does it mean for problem X to be at least as difficult as problem Y?

Suppose? no, we may make it harder or easier.
Use some to Y as an input to X:
what if there are multiple ways to solve X, some using Y and some not?
A first notion of reduction is the following, called “Turing reduction” (after Alan Turing). We say problem Y can be “Turing reduced” to X if there is a polynomial time algorithm for Y making no more than a polynomial number of calls to a black box for solving X. Then, intuitively, because X can be used to solve Y in this way, it must be at least as difficult as Y.
4. While the notion of Turing reduction is natural, and used in several contexts, the relevant notion for the purposes of this class (and the theory of NP-completeness) is somewhat different. The terminology that will be frequently used is: Y reduces to X in polynomial time, which we denote by \( Y \leq_p X \). The idea is that in polynomial time, from any input \( y \), we can compute an input \( x = f(y) \) such that the correct answer for \( y \) (as an instance of Y) is “Yes” if and only if the correct answer to \( x \) (as an instance of X) is “Yes”. This type of reduction is sometimes called a “Karp reduction” (after Richard Karp), but more frequently, we simply call it a polynomial-time reduction, without the need to distinguish it from Turing reductions.

5. To further explain the above notion of reduction, a diagram was provided, indicating the reduction steps as well as the yes/no requirements. The requirement is that yes/no instances map to yes/no
instances. The important distinction from Turing reductions is that this type of reduction only gets to use the black box once, at the very end, and has to give exactly the same output.

6. Next, we looked at three prototypical problems in NP. Notice that in keeping with standard notation, the name of a problem will be written in all capitals. Thus, we can distinguish the CLIQUE problem, which is to find a set of size at least k such that every pair of nodes in the set is directly connected by an edge, from a clique, which is the actual set with those properties.

The INDEPENDENT SET problem was introduced next. We are
looking for the largest “un-connected” subset (no two nodes in the subset have an edge between them). The INDEPENDENT SET problem is frequently useful for modeling problems where entities have conflicts, and we want to select as many as possible while avoiding any conflicts within the selected set. For example, if nodes represent projects, and an edge between two nodes means there is a conflict (both need the same computer or human resource) between these two projects. Our goal is to pick up as many non-conflicting projects as possible. Similarly, if nodes are courses, then edges could be schedule overlaps, and the goal would be to select as many courses as possible without overlaps. Or nodes could be friends, edges could be animosities, and the goal is to invite as many friends as possible while avoiding fights.

The input of the VERTEX COVER problem is a graph G and a number k. Our goal is to find a subset S of at most k nodes, such that each edge in G has at least one endpoint in S. Real life examples frequently involve “monitoring edges” in some sense. For example, the problem of placing emergency phones or police officers at intersections, such that each road is viewed by an officer from at least one of its ends, or can see an emergency phone at at least one endpoint. Similarly, we could be interested in tapping as few phones as possible, while ensuring to monitor all conversations between suspicious pairs of individuals.
Among the above three problems, first we are going to prove that INDEPENDENT SET \( \leq_p \) CLIQUE. Thus, we need to design a polynomial time function \( f: (G,k) \rightarrow (G',k') \) such that:

1. If \( G \) has an independent set of size at least \( k \), then \( G' \) has a clique of size at least \( k' \).

2. If \( G \) has no independent set of size at least \( k \), then \( G' \) has no clique of size at least \( k' \).

A function that works for this reduction is the function \( f \) that maps graph \( G \) to its complement graph \( G' = (V, E^c) \) and sets \( k' \) to be equal to \( k \). Here \( E^c \) means taking the complement of the original edge set \( E \). That is, if \( G \) had an edge between \( u \) and \( v \), then \( G' \) does not have an edge between \( u \) and \( v \), and vice versa.
8. Function $f$ can certainly be computed in polynomial time, since it is just flipping all entries in the adjacency matrix. To prove that the reduction is correct, we prove:

(1) If $(G, k)$ is a “yes” instance, then $G$ has an independent set of size at least $k$. After flipping each edge, the same set, which was formerly an independent set (un-connected), now becomes a clique (fully connected set). Therefore, $(G', k')$ is a “yes” instance of the CLIQUE problem.

(2) For the “no” instance part, we can equivalent show that if $(G', k')$ is a “yes” instance for the CLIQUE problem, then $(G, k)$ is a “yes”
instance for INDEPENDENT SET. This is equivalent to proving that if \((G, k)\) is a “no” instance, so is \((G', k')\). The proof for this direction is nearly identical to the original one: if \((G', k')\) is a “yes” instance, then \(G'\) must contain a clique \(S\) of size at least \(k' = k\). Thus, the same set \(S\) is independent in \(G\), proving that \(G\) has an independent set of size at least \(k\), and is thus also a “yes” instance.

Now, we have proved that finding large clique in a given graph is at least as difficult as finding a large independent set in a given graph. With exactly the same function for reduction, we can also prove that

\[ \text{CLIQUE} \leq_p \text{INDEPENDENT SET} \]

Thus, we have shown that either both the CLIQUE problem and the
INDEPENDENT SET problem are polynomial time solvable, or neither of them is.

But if $S$ is a clique in $G$, then $S$ is an independent set before flipping in $H$. So $S$ has an independent set of size at least $k$, so it is a "Yes" instance. Do So, finding both cliques $k$ at least as difficult as finding large independent sets.

We exactly the same reduction, nearly identical proof.