CS 3510: Design & Analysis of Algorithms 11/06/2023

Homework 8: NP-completeness II

Prof. Abrahim Ladha Due: 11/12/2023 11:59pm

- **Please TYPE your solutions using Latex or any other software. Handwritten solutions *won't* be accepted.**

Steps to write a reduction

To show that a problem is NP-complete, you need to show that the problem is both in NP and NP-hard. In a polynomial-time reduction, we have a *Problem B*, and we want to show that it is NP-complete. These are the steps:

- Demonstrate that *problem B* is in the class NP. This is a description of a procedure that verifies a candidate's solution. It needs to address the runtime. You should give a polytime verifier which takes as input a candidate problem, and a witness, and outputs true or false if the witness is a solution.
- Demonstrate that *problem B* is at least as hard as a problem previously proved to be NP-complete. Choose some NP-complete A and prove $A \leq_p B$. You need to show that *problem B* is NP-hard. This is done via polytime reduction from a known NP-complete *problem A* to the unknown *problem B* ($A \to B$) as follows:
  1. Choose A. You will have many NP-complete problems to pick from later on, and it is best to pick a similar one.
  2. Give your reduction f and argue that it runs in polynomial time.
  3. Prove that $x \in A \implies f(x) \in B$
  4. Prove that $x \notin A \implies f(x) \notin B$
  5. This is sufficient to show $x \in A \iff f(x) \in B$. Since A was NP-complete, and $A \leq_p B$, we can conclude that B is NP-hard. Note: You don't have to provide a formal proof. You can briefly explain in words both implications and why they hold.

The second bullet point above is prove that *problem B* is NP-hard which combined with the first bullet point yields the NP-complete proof.

Problem 1

(25 points)

The `Almost-SAT` problem takes as input a boolean formula on n literals, in conjunctive normal form with m clauses. The output is an assignment of the literals such that **exactly** $m-1$ clauses evaluates to `TRUE`, if such assignment exists, and outputs NO otherwise. Show that `Almost-SAT` is NP complete.

Solution:

For showing that Almost-SAT is in NP, we can do this in polynomial time by going through each clause and checking whether the clauses are satisfied, and hence verify if the literals satisfy the formula.

We will now reduce SAT to Almost-SAT: Given an input to SAT, with m clauses and n variables, we simply construct a new input to Almost-SAT, with $m+2$ clauses and n variables. Specifically, we add the clauses (x_1) and $(\neg x_1)$. Then, there's an almost-assignment for this new input if and only if there's an assignment for our original input for SAT. We show this correctness in both directions:

($\implies$) If Almost-SAT has an almost-assignment, then either the (x_1) or $(\neg x_1)$ clause must be the one that isn't satisfied, since both cannot be satisfied at the same time. Thus, if we remove both of these clauses, everything that remains must be satisfied, and we have an assignment for SAT.

($\impliedby$) If SAT has an assignment, then in that assignment, x_1 must either be true or false. If it's true, then the extra (x_1) will be satisfied, but the $(\neg x_1)$ won't, and likewise, if it's false, only the extra $(\neg x_1)$ will be satisfied. Using the same assignment in our new input for Almost-SAT, the first m clauses will all be satisfied, and one of the last 2 clauses won't, meaning we have an almost-assignment for Almost-SAT.

Since the reductions can be done in polynomial time, this proves that Almost SAT is NP-complete, as desired.

Problem 2
(25 points)

The `Dense Subgraph Problem` takes as input an undirected graph $G = (V, E)$ and two positive integers a and b, and returns a subset of *exactly* a vertices such that there are *at least* b edges connecting them, if such set exists, or return NO otherwise.

Show that the `Dense Subgraph Problem` is `NP-complete`.

Solution: First, we can start be showing that the `Dense Subgraph Problem` $\in$ NP. Given a subset of a vertices, we can count the number of edges connecting them in $\mathcal{O}(|E|)$ time to verify that the solution has at least b edges connecting the vertices. Therefore the `Dense Subgraph Problem` $\in$ NP.

Now, we can use the NP-Complete problem of `Clique-search` to use a proof by generalization and form a reduction. Consider an instance I of `Clique-search` $= (G, g)$ where G is an undirected graph and g is the number of vertices. We can transform this into an instance $f(I)$ of the `Dense Subgraph Problem` by setting $a = g$ and $b = \frac{1}{2}(g)(g-1)$. Since a clique is fully connected, it must contain $\binom{g}{2} = \frac{1}{2}(g)(g-1)$ edges. Similarly, any subgraph which has $\frac{1}{2}(g)(g-1)$ edges connecting all g vertices must be a clique, since it is the maximal number of edges in a subgraph, and so every vertex is adjacent to the others.

Therefore if I has a solution in `Clique-search`, then $f(I)$ in the `Dense Subgraph Problem` must have a solution since a clique is a subgraph with $\frac{1}{2}(g)(g-1)$ edges.

Similarly, if $f(I)$ has a solution in the `Dense Subgraph Problem`, we can use the property of a clique to prove that a subgraph of g vertices with $\frac{1}{2}(g)(g-1)$ edges must be a clique. Therefore there is a solution of instance I in `Clique-search` if and only if there is a solution of the instance $f(I)$ in the `Dense Subgraph Problem`. We recover a solution to `Clique-search` from the `Dense Subgraph Problem` in polynomial time, since the solution will be exactly the same from the instance of the `Dense Subgraph Problem`.

Therefore, since we can express `Clique-search` as a specific case of the `Dense Subgraph Problem`, by generalization we have that the `Dense Subgraph Problem` is NP-complete. □

Problem 3
(25 points)

Show that the following CLIQUE-&-INDEPENDENT-SET problem is NP-Complete.

Input: an undirected graph $G = (V, E)$ and an integer k.
Output: Returns both a $k-$clique and $k-$independent set in G i.e. a set $Q \subseteq V$ of k vertices that are fully connected, AND another set $R \subseteq V$ of k vertices with no edges between them. Sets R and Q might not be disjoint.

Note: Write the reduction in the format described on the front page of this document.

Solution:

Problem is in NP: We simply check that there are k vertices in each set returned, and then verify that all possible edges are present in one of them, and no edges are present in the other, which takes at most $\mathcal{O}(n^2)$ each. Thus, CLIQUE-&-INDEPENDENT-SET is in NP.

Choose NP-Complete problem Clique. Given an input (G, k) for Clique, construct G' by adding k new isolated vertices, effectively adding a $k-$sized independent set to the graph. This is a reduction that takes $\mathcal{O}(k)$ time which is therefore polynomial.

$\phi \in$ Clique $\implies f(\phi) \in$ CLIQUE-&-INDEPENDENT-SET
If I has a solution in Clique, then a clique of size at least k exists in graph G. This means that G' has the same clique, since we kept all the same vertices, and it also has the $k-$sized independent set we added, I' has a solution.

$\phi \notin$ Clique $\implies f(\phi) \notin$ CLIQUE-&-INDEPENDENT-SET
Showing the contrapositive: if I does not have a solution in Clique, then a clique of size at least k doesn't exists in graph G. Since we are keeping all the same vertices and edges from graph G, I' won't have both a $k-$sized clique and $k-$sized independent set in G'. Hence, I' does not have a solution as well.

This proves CLIQUE-&-INDEPENDENT-SET is NP-Hard.

Since CLIQUE-&-INDEPENDENT-SET is both in NP and is NP-hard, it is NP-complete.

Problem 4
(25 points)

In the `Special-Two-Sum` problem, you're given a set of integers S, and you attempt to split the set up into two non-overlapping sets A and B such that $A \cup B = S$, $A \cap B = \emptyset$ and

$$\sum_{a \in A} a = \sum_{b \in B} b$$

. Show that `Special-Two-Sum` is `NP`-complete.

Solution: This problem is in NP because we can do a verification by checking the equality the two of the sets in polynomial time.

We will reduce from SubsetSum. Given a set S and target t for SubsetSum, we will will create the set $S' = S \cup s - 2t$, where s is the sum of S, and pass that into `Special-Two-Sum`.

If there's a solution to SubsetSum, this implies there's some set of numbers which add to t. The sum of the other numbers will be $s - t$; since we added $s - 2t$, if we add this number to the side that sums to t, it will equal $s - t$, meaning there are two equal sized sets. If there's no valid solution to SubsetSum, this proof will not work. Then we know it's NP-complete.