CS 3510: Design & Analysis of Algorithms 10/29/2023

Homework 7: NP-completeness I

Prof. Abrahim Ladha Due: 11/05/2023 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

(30 points)

Show that each of the following problems is in NP. For each problem, assume the witness gives you a candidate solution S. You must show how to verify S in polynomial time.

(a) Given an array of n distinct integers, return those integers sorted in descending order.

Solution: Iterate through S, check if it is in sorted order by comparing the next element to the current element. Keep track of all integers found and ensure there are no missing or extra integers from the input. This takes $O(n)$ time, which is polynomial.

(b) Given a directed acyclic graph $G = (V, E)$, return a valid topological sorting of the graph.

Solution: Check that each vertex in G is represented in the topological ordering S. Iterate through each edge in the graph and check that it points forward in S. This takes $O(n)$ time, which is polynomial.

(c) Given a list of sets $\{S_1, S_2, \ldots, S_n\}$ and a budget b, return a set H of size $\leq b$ which intersects every S_i.

Solution: Check if H is of length $\leq b$. Then, iterate through each set S_i and check that $S_i \cap H \neq \emptyset$. This takes $O(\max(|S_i|) \times b)$, which is polynomial.

Problem 2

(20 points)

Show that the following `All-trueORfalse` problem is in class P.

Input: A boolean formula in CNF with m clauses and n variables such that each clause C_i has exactly three literals.
Output: An assignment such that each of the three literals in every clause C_i, for $1 \leq i \leq m$, are either all True or all False.

Solution:

Here are three possible solutions:

1. Graph of implications. Create an undirected graph of implications from 2-SAT where there exists edges between each of the three literals in a clause. Run DFS on the graph to identify connected components. For each connected component, set all literals to True or False as needed. If there is a connected component with both literals for a given variable (e.g. x_1 and $\overline{x}_1$), there is no possible assignment giving all True or False literals.
2. Greedy solution: assign $x_1 = 1$. Loop through all clauses, and for those containing x_1 assign 1 to all those literals, and for those containing $\overline{x}_1$, assign 0 to all those literals. Continue assigning all variable until you reach a contradiction. In the second case, repeat but assigning $x_1 = 0$ instead of 1.
3. Convert each clause to three clauses using this pattern: $(x_1 \vee x_2 \vee x_3) \rightarrow (x_1 \vee \overline{x}_2) \cap (x_2 \vee \overline{x}_3) \cap (x_3 \vee \overline{x}_1)$; to satisfy these three clauses each variable will be set to the same value (T or F). Pass this converted CNF to `2-SAT` which solves it as a black box. It provides either a truth assignment or outputs None, which we return to `All-trueORfalse`.

Problem 3
(20 points)

Answer the following questions using proofs when required or explanations:

(a) Prove that polynomial-time reductions are transitive. That is, if $A \leq_p B$, and $B \leq_p C$, then $A \leq_p C$.

Solution: If $A \leq_p B$, then we can transform an input of A into an input of B in polynomial time, and then convert the solution of B back into a solution of A in polynomial time. Likewise, if $B \leq_p C$, we can convert an input of B into an input of C in polynomial time, and convert the solution of C back into B in polynomial time. Therefore it must be true that we can take an input of A, transform it to B, and then to C in polynomial time, and then convert the solution from C back into one from B, back into one from A also in polynomial time. Therefore $A \leq_p C$.

(b) Explain why polynomial-time reductions are not symmetric. That is, if $A \leq_p B$, it is not necessarily true that $B \leq_p A$. Give an example of when it is true.

Solution: If $A \leq_p B$, then we know B is at least as hard as A, but we cannot conclude that A is as hard as B. For example, A can be `2-SAT` and B can be `SAT`. However, this would be symmetric if A and B are both problems in P, or NP-Complete problems.

(c) True or False: If $A \leq_p B$ and there is an algorithm for B in $O(n^3)$ time, must A be solvable in $O(n^3)$?

Solution: False. If the reduction takes $O(n^7)$.

Problem 4
(30 points)

Show that the following `TwoThirdsMajority-SAT` problem is NP-Complete.

Input: a boolean formula in CNF with n variables and m clauses, where m is divisible by 3.
Output: A satisfying assignment, if exists, such that at least 2/3 of the clauses in the given input CNF evaluates to True.

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

Solution:

Problem is in NP: Suppose we have a candidate assignment S to `TwoThirdsMajority-SAT`. We can simply iterate through each clause and apply the assignment, and record whether it was satisfied or not in a counter. At the end of iteration, if the number of satisfied clauses is more than $\frac{2m}{3}$, the solution is correct. This takes $\mathcal{O}(nm)$ time, which is polynomial. Therefore this problem is in NP.

Choose NP-Complete problem `SAT`. Our reduction from `SAT` to `TwoThirdsMajority-SAT` is as follows: given input I for `SAT`, we can generate input I' for `TwoThirdsMajority-SAT` by adding m clauses (y), where y is a new literal, as well as m clauses of $(\overline{y})$. For instance if the original input is $(x_1 \vee x_2) \cap (x_2 \vee x_3)$ with $m = 2$, a clause from the input I is transformed as follows:

$$(x_1 \vee x_2) \cap (y) \cap (\overline{y}) \cap (x_2 \vee x_3) \cap (y) \cap (\overline{y}) \text{ with } m = 6.$$

Therefore I' has $3m$ clauses where m clauses are the original clauses from I. This is a reduction that takes $\mathcal{O}(m)$ time which is therefore polynomial.

$\phi \in$ `SAT` $\implies f(\phi) \in$ `TwoThirdsMajority-SAT`
If I has a solution in `SAT`, then the first m unmodified clauses all evaluate to True. One set of clauses (y) or $(\overline{y})$ must all evaluate as True since they use the same literal. Since the original input is True as well, $2m$ clauses must be True. As a result, at least $2m$ clauses evaluate to True in I', so `TwoThirdsMajority-SAT` must have a solution.

$\phi \notin$ `SAT` $\implies f(\phi) \notin$ `TwoThirdsMajority-SAT`
Showing the contrapositive: if I does not have a solution in `SAT`, then at least one of the first m clauses must evaluate to False by definition. It is impossible for more than m clauses to evaluate to True in the additional clauses added to I', so the number of satisfied clauses must be less than $2m$ in I'. This implies that there is no two-thirds majority, so there is no solution for `TwoThirdsMajority-SAT` on I'.

This proves `TwoThirdsMajority-SAT` is NP-Hard.

Since `TwoThirdsMajority-SAT` is both in NP and is NP-hard, it is NP-complete.