Vivek Pandey

vpandey35@gatech.edu

Problem 1 (Almost SAT)

References:

Joves Notes

Approach:

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 evaluate to TRUE, if such assignment exists, and outputs NO otherwise.

In order to prove that Almost-SAT is NP-complete, I will show that

- A solution to Almost-SAT can be verified in polynomial time such that $Almost\text{-}SAT \in NP$
- A known NP-complete problem, the SAT problem, can be reduced to an Almost-SAT problem i.e., $SAT \rightarrow Almost\text{-}SAT$

NP Proof:

For a given input I to the Almost-SAT problem, we look for a solution S.

- If a solution doesn't exist, we return NO.
- If a solution does exist, we can take the T/F assignments from S and plug-them into each clause of I to verify the solution. Since it is O(n) to check each clause, the runtime is O(mn) to verify the entire conjunctive form with m clauses. Since this is polynomial time, we have proven that $Almost\text{-}SAT \in NP$

Input: ($SAT \rightarrow Almost\text{-}SAT$)

Let's take the input **i** to the SAT problem in a conjunctive normal form (CNF), with **n** literals and **m** clauses. Now let's create a new variable **x** and transform the input such that $i' = i \cap x \cap \bar{x}$. This new CNF **i'**, which now has $n + 1$ variables denoted by **n'** and $m + 2$ clauses denoted by **m',** is the input to Almost-SAT.

Runtime for this part is O(1) since we are just adding two constant clauses.

Output: ($SAT \rightarrow Almost\text{-}SAT$)

If Almost-SAT returns NO, return NO for SAT.

Otherwise, return the Almost-SAT solution S by setting x=True and evaluating i' CNF.

Runtime for this O(n) to assign values for n variables.

Correctness:

We can show the correctness of this reduction by showing that:

$$i \text{ is satisfied} \Leftrightarrow i' \text{ is satisfied with } m' - 1 \text{ clauses}$$

First, we show the forward implication i.e., if i is satisfied then i' is satisfied with m'-1 clauses. Since x is set to True in i' which implies the clause $\bar{x}$ evaluates to False, if i is satisfied, then i' is satisfied with m'-1 clauses.

Next, we show the reverse implication i.e., if i' is satisfied with m'-1 clauses, then i is satisfied. No matter whether we set the value of x to True or False, one of the clauses out of $x \cap \bar{x}$ will always come true, and the other will always come false. Therefore, it has no impact on the outcome of other clauses. Therefore, when i' is satisfied with m'-1 clauses, then i is guaranteed to be satisfied.

<u>Problem 2 (Clique-IS)</u>

References:

Joves Notes

Approach:

Given an undirected graph G = (V, E) and an integer k, the Clique-IS problem returns a clique of size k as well as an independent set (IS) of size k, provided both exist.

In order to prove that this Clique-IS problem is NP-complete, I will show that

- A solution to Clique-IS problem can be verified in polynomial time such that $\text{Clique-IS} \in NP$
- A known NP-complete problem, the Clique problem, can be reduced to a Clique-IS problem i.e., $\text{Clique} \rightarrow \text{Clique-IS}$

NP Proof:

For a given input graph G = (V, E) and integer k to the Clique-IS problem, we look for a solution consisting of a clique C and Independent Set S.

- If a solution doesn't exist for either C or S, we return NO.
- If a solution does exist, we need to verify it is valid solution by checking the following three things:
  - |C| == |S| == k. O(1) time complexity
  - C is a valid clique i.e., for all pairs of vertices (v, w) in C, an $edge\ \mathrm{e} = (v, w) \in E$. O(n^2) time complexity.
  - S is a valid independent set i.e., for all pairs of vertices (v, w) in S, an $edge\ e = (v, w) \notin E$. O(n^2) time complexity.

Since we can verify the solution of Clique-IS problem in polynomial time, we have proven that $\text{Clique-IS} \in NP$

Input: $(\text{Clique} \rightarrow \text{Clique-IS})$

We take the input to the Clique problem - an undirected graph G = (V, E) and integer k. Now we create another graph G' = (V', E) which is a copy of G but with additional vertices such that V' = V + S where S = Independent set of G with |S| = k. No new edges will be added to G'. Therefore, the new graph G' can be created by copying G in O(|V| + |E|) time and adding new vertices S in O(|V|) time. Hence, the runtime remains polynomial in original input size.

Output: $(\text{Clique} \rightarrow \text{Clique-IS})$

If Clique-IS returns NO, return NO for Clique.

Otherwise, Clique-IS solution consists of a Clique C and Independent Set I as the outputs with equal size k. Return only the Clique C computed while solving the Clique-IS problem on G'. Since no edges were added, the Clique C for G' will also be the Clique for G.

We will also drop the independent vertices from G' with no edges, which yields the original graph G since no edges were added. Runtime for this removal is O(|V|) since |I| = k <= |V|.

Correctness:

We can show the correctness of this reduction by showing that:

$$\text{C is a clique in G} \Leftrightarrow \text{C is a clique in G'}$$

First, we show the forward implication i.e., if C is a clique in G, then C is a clique in G’. Since graph G and G’ have the same set of edges, the vertices that meet the condition for being added to a clique in G are the same vertices that will be added to the clique of G’.

Next, we show the backward implication i.e., if C is a clique in G’, then C is a clique in G. The only additional vertices that exist in G’ that are not in G are the independent set vertices which cannot be a part of any clique. And the number of edges is the same in G’ and G. Therefore, the clique computed for G’ will be the same as the clique for G.