CS 3510: Design & Analysis of Algorithms October 28th 2024

Problem Set 6: NP-Completeness

Jinseo Lee Due: November 4th 2024

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- Your algorithms must be in plain English & mathematical expressions, and the pseudo-code is optional. Pseudo-code, without sufficient explanation, will receive no credit.
- If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.
- Unless a question explicitly states that no work is required to be shown, you must provide an explanation or justification for your answer

Problem 1: MST in NP

(20 points)

In this problem, you shall write a verifier to demonstrate that the decision variant of the Minimum Spanning Tree (MST) problem belongs to the class NP. Consider a graph $G = (V, E)$ with a set of vertices V and edges E, along with a weight threshold k. The decision variant of MST asks: Is there a spanning tree of G with a total weight less than or equal to k?

To write a verifier for this problem, you are given a subset of edges $T \subseteq E$. Your task is to determine whether T is a valid minimum spanning tree of G with a weight $\leq k$.

Provide an efficient algorithm that performs this verification and a short justification to support your answer.

Solution:

In order to verify whether T for the graph G possess a total weight less than or equal to k, first thing we need to check is to determine whether T fulfills the three key components of MST characteristics: minimum, spanning, and a tree. This is because the decision variant will return true if and only if the total weight of MST is less than or equal to k as MST is the tree with the smallest possible total weight within a given graph. Therefore, the algorithm considers the following conditions:

1. T is a valid spanning tree: We can run Depth-First-Search on T to check whether T connects up the graph G (spanning) and T has no cycles (tree). Either if it does not fully connect up G or it has the cycle, it is false. More specifically, you can check for the connectivity by exploring the visited vertices and check if it contains all vertices of G, after the DFS iteration. You can check for the acyclicity by: For each neighbor v of u, if v is already visited and v is not the parent of u , then a cycle is detected. Overall, they could be determined after one iteration of DFS. Since T must have $|V| - 1$ edges in order to be spanning, the runtime equals $O(|V|)$ as DFS takes $O(|V|+|E|) = O(|V|+|V|-1) = O(2|V|) = O(|V|)$.

2. T has the minimum weight: We can run Prim's algorithm on G to identify the weight of MST. Then, we check if T has the same weight from what we have found with the Prim's algorithm. If the total weight of the MST and that of T are the two different values, it is false. Overall runtime for step 2 equals that of Prim's algorithm, which is $O((|E|+|V|)log|V|)$.

3. Assuming that T satisfies all of the previous conditions, we now know that T is a valid MST given a graph G. Therefore, we can simply compare the total weight of T from what we have found in '2' and compare it with k. If it is larger, it is false.

The total runtime for the verification algorithm is $O((|E| + |V|)log|V|)$ (it dominates $O(|V|)$), which is bounded by the polynomial time. Therefore, this MST verification is in NP.

Problem 2: Shortest Path Reduction

(40 points)

In the standard shortest path problem, you are given a graph $G = (V, E)$ with weights w_e on each edge. As we have learned in this course, there are efficient algorithms to solve this problem when the weights are positive. In the node-weighted variant of the shortest path problem, you are given a graph $G = (V, E)$ with weights w_v assigned to each node $v \in V$. We would like to demonstrate that the standard shortest path problem with edge weights is at least as hard as this node-weighted version.

1. Given an instance of the node-weighted shortest path problem, show how to reduce it to an instance of the standard shortest path problem with edge weights. Specifically, construct a weighted graph $G' = (V', E')$ and define edge weights w'_e in such a way that finding the shortest path in G' corresponds to finding the shortest path in G.

   Solution:

   Let G be a directed graph with node weights, and G' be a graph with edge weights. We first introduce V_{in} and V_{out}, the two new variables for every vertex in G'. We can reduce the node weight problems to the edge weight problems by re-interpreting W_v, the weight of the node v, as the sum of the weight of in-degree and out-degree (i.e $W_v = V_{in} + V_{out}$). More specifically, these two variable represent the total cost of traversing the corresponding vertex in G', which is same as the weight of such a vertex in the original graph G. Second step during the reduction is to connect the disconnected edges: For each edge (u,v) in the G', add an edge (u_{out}, v_{in}) with weight 0. This setup allows traversal between nodes in G' without adding extra weight beyond the nodes' weights.

   For instance, let A and B the vertex of G with weights W_A and W_B. Then, G' would look like A_{in} - A - A_{out} - B_{in} - B - B_{out}. Edges (A_{in},A_{out}) equal to W_A, edges (B_{in},B_{out}) equal to W_B, and (A_{out}, B_{in}) doesn't have any weight.

   The runtime for this reduction is proportional to the number of vertices and edges in G as adding the edges weight of 0 that connects u_{out} and v_{in} takes O($|E|$) time. Also, we add two additional edges, v_{in} and v_{out}, for every vertex, which makes the runtime O($2|V|$) = O($|V|$) time. Therefore, the overall runtime is O($|V| + |E|$), which is in polynomial time.

2. Prove that the shortest path in the edge-weighted graph G' will have a weight equal to the weight of some path in the node-weighted graph G.

   Solution:

   Suppose any path P in G from the vertex s to the vertex t. The total weight of the corresponding path could be represented in G' by traversing the following edges: $s_{in}, s_{out}, v_{in}, v_{out}, ..., t_{in}, t_{out}$. In G, we can calculate the total weight of P with the following equation: $\sum_{v \in P} w_v$. In G', we can calculate the same thing with the follow-

ing equation: $\sum_{v \in P} v_{in} + v_{out}$. By construction of G', we know that w_v equals $v_{in} + v_{out}$.

We can conclude that any path in G from s to t can be represented in G' with an identical weight, showing that every path weight in G has a corresponding path in G' with the same weight. Therefore, the shortest path in G' will have a weight equal to the weight of some path in G, as required.

3. Prove that the shortest path in the edge-weighted graph G' corresponds to the shortest path in the node-weighted graph G.

Solution:

Since (node weighted algorithm) $\rightarrow$ (edge weighted algorithm) is what we'd like to demonstrate, we define f to be the function of finding shortest path in the node weighted graph G and f' to be that of edge weighted graph G'. With G' defined in 2.1, we can derive f' by introducing v_{in} and v_{out} to transform the input of G to that of G'. This time, we define o to be the output of (node weighted algorithm) and o' to be that of (edge weighted algorithm). We can find o', which is the shortest path of G', by running Dijkstra's algorithm. We have shown in 2.2 that any path in G' is structurally and weight-wise identical to the corresponding path in G, proving that the shortest path in G' corresponds exactly to the shortest path in G. Therefore, the shortest path we have found with one run of Dijkstra's on G' would be identical to that of G.

Problem 3: 33SAT to 3SAT

(40 points)

The 33SAT problem is like the 3SAT problem. The only difference is that in this problem, each variable appears in no more than 3 separate clauses. The problem is to decide if 33SAT is satisfiable.

1. Prove that 33SAT is in the class NP.

   Solution:

   In order for 33SAT to be in the class NP, it must be verified in polynomial time. We can verify the given candidate solution by assigning the True or False to the variables, which overall takes constant time for each clause as there are at most three literals in each of them. Given 33SAT problem with m clauses and n variables, verifying the solution will take O(m) time as a whole since there are m clauses and each of them takes O(1) time. Therefore, 33SAT is in the class NP.

2. Provide a reduction that demonstrates that 33SAT is NP-HARD.

   Solution:

   To demonstrate that 33SAT is NP-HARD, we need to reduce 3SAT to 33SAT by referring to the following property: SAT, which is guaranteed to be in NP-hard, can be reduced to an equivalent instance of 3SAT. Since 3SAT is in NP-hard, 33SAT is NP-hard if and only if 3SAT can be reduced to 33SAT.

   Suppose an instance I of 3SAT with at most 3 literals of a_i $(1 \leq i \leq n)$ for m clauses, and certain variable a_i appears in $k > 3$ clauses. To transform I into I', an instance of 33SAT, we can replace its first appearance by x_1, second appearance by x_2, and so on. Finally, add the following clauses: $(\bar{x_1} \cup x_2) \cap (\bar{x_2} \cup x_3) \cap ... \cap (\bar{x_k} \cup x_1)$. Here, these clauses ensure that if any x_i is true, the others will follow through the auxiliary variables. And repeat this for every variable that appears more than three times.

   With the transformation of I to I' above, we ensure that each clause maintains at most 3 literals and no literals appear more than 3 times. Since the output for the input I' satisfies the 33SAT algorithm, we can claim that the output for I also satisfies the 3SAT algorithm. Also, reduction takes O(m) time identifying the variable occurrences, renaming variables, and adding clauses. Hence, we can reduce 3SAT to 33SAT in poly-time, demonstrating the 'NP-Hard'ness of 33SAT.

3. Prove that your reduction is correct by showing that it maps "yes" instances of the original problem to "yes" instances of 33SAT and "no" instances to "no" instances.

   Solution:

   "Yes": Suppose the original 3SAT instance is satisfiable. Then, there exists an assignment of truth values to variables that satisfies all clauses. After the reduction, each copy of a variable in the 33SAT instance will take the same truth value as the original

variable due to the synchronization clauses. Since each clause in 33SAT contains the appropriate copies of variables (with no more than three appearances), the satisfying assignment from 3SAT translates directly to a satisfying assignment in 33SAT.

"No": If the original 3SAT instance is not satisfiable, no assignment can satisfy all clauses simultaneously. The reduction only introduces additional clauses to enforce consistency among variable copies without changing the logical structure of the formula. As a result, if the original instance is unsatisfiable, the transformed 33SAT instance will also be unsatisfiable because no assignment can simultaneously satisfy all clauses.