CSEP531 Homework 4 Solution

1. **DOUBLE-SAT is in** NP. The polynomial size certificate consists of two assignments f_1 and f_2. First, the verifier verifies if $f_1 \neq f_2$. Then, it verifies if both assignments satisfy ϕ by subtituting the values for the variables and evaluate the clauses of ϕ. Both checks can be done in linear time.

   DOUBLE-SAT is NP**-hard**. We give a reduction from SAT. Given an instance ϕ of SAT which is a CNF formula of n variables $x_1, x_2, \ldots x_n$, we construct a new variable x_{n+1} and let $\psi = \phi \wedge (x_{n+1} \vee \neg x_{n+1})$ be the corresponding instance of DOUBLE-SAT.

   We claim that ϕ has a satisfying assignment iff ψ has at least two satisfying assignments. On one hand, if ϕ has a satisfying assignment f, we can obtain two distinct satisfying assignments of ψ by extending f with $x_{n+1} = T$ and $x_{n+1} = F$ respectively. On the other hand, if ψ has at least two sastisyfing assignments then the restriction of any of them to the set $\{x_1, x_2, \ldots x_n\}$ is a satisfying assignment for ϕ.

   Thus, DOUBLE-SAT is NP-complete.

2. k**-SPANNING-TREE is in** NP. The polynoial size certificate is a subgraph of G. The verifier needs to verify that this subgraph

   (a) is connected;

   (b) is acyclic;

   (c) spans all the vertices of G; and

   (d) has maximum degree at most k.

   The first three conditions can be checked by a depth-first traversal in which we mark a vertex when we visit it for the first time. If we ever visit a marked vertex, the subgraph contains a cycle; otherwise, it is acyclic. If all the vertices of G are marked by the traversal, then the subgraph is connected and spans G. Otherwise, either it is not connected or it doesn't span G. Finally, checking for the maximum degree involves going through all the vertices and count theirs neighbors. All these checks can be done in time linear in the number of edges of the subgraph.

   k**-SPANNING-TREE is** NP**-hard**. First, note that 2-SPANNING-TREE is exactly undirected HAMPATH: a tree in which each vertex has degree at most 2 is a path. Thus, 2-SPANNING-TREE is NP-hard.

   Now, we give a reduction from 2-SPANNING-TREE to k-SPANNING-TREE for any $k \geq 3$. Given a graph $G(V, E)$ as an instance of 2-SPANNING-TREE, we construct another graph $G'(V', E')$ where V' contains V and some other vertices as follows. For each vertex $v \in V$, V' also contains $k-2$ new vertices $v_1, v_2, \ldots v_{k-2}$. Similarly, E' contains E and some extra edges connecting v_i and v. Formally,

$$\begin{aligned} V' &= V \cup \{v_1, v_2, \ldots v_{k-2} | v \in V\} \\ E' &= E \cup \{v_1 v, v_2 v, \ldots v_{k-2} v | v \in V\}. \end{aligned}$$

   We claim that G has a 2-SPANNING-TREE iff G' has a k-SPANNING-TREE. On one hand, assume that G has a 2-SPANNING-TREE T, then $T' = T \cup \{v_i v | v \in V, i = 1, 2, \ldots k-2\}$ is a spanning tree of G'. Since each vertex of T has degree at most 2, each vertex of T' has degree at most k.

   On the other hand, assume that G' has a k-SPANNING-TREE T'. One can see that T' must contains all the edges $v_i v$, since those are the only edges incident at v_i. Furthermore, all v_i's are

leaves of T' since they all have degree 1. Thus we can remove them to obtain a spanning tree T of G. Since removing all edges $v_i v$ decrease the degree of each v by $k-2$, T is a 2-SPANNING-TREE of G.

Thus, k-SPANNING-TREE is NP-complete for any $k \geq 2$.

Note: you can also do a reduction from k-SPANNING-TREE to $(k+1)$-SPANNING-TREE for any $k \geq 2$.

3. **MINES-CONSITENCY is in** NP. The polynomial size certificate is a placement of mines. Checking such a certificate involves walking through all the vertices in the graph and for each vertex, count the number of mines around it. This can be done in $O(|E|)$.

   MINES-CONSISTENCY is NP**-hard**. We give a reduction from 3SAT. Given a formula ϕ on n variables $x_1, x_2, \ldots x_n$ as an instance of 3SAT, we construct an instance G of MINES-CONSISTENCY as follows.

   For each variables x_i, create two unlabeled vertices corresponding to x_i and $\neg x_i$. We will abuse the notations and use x_i and $\neg x_i$ to name these two vertices. Then, we create a vertex t_i labeled by 1 and connect it to both x_i and $\neg x_i$. This guarantees that a bomb is placed at either x_i or $\neg x_i$ but not both, correspond to the fact that exactly one of x_i and $\neg x_i$ is true.

   For each clause C, we create 3 extra vertices u_C, v_C and h_C where both u_C and v_C are unlabeled while h_C is labeled by 3. We then connect u_C and v_C to h_C. Finally, assume that $C = l_i \vee l_j \vee l_k$ where l_i, l_j and l_k are literals, i.e., either variables or their negation, we connect l_i, l_j and l_k to h_C. Clearly, this reduction can be done in polynomial time.

   We claim that ϕ has a satisfying assignment iff G has a consistent placement of mines. One one hand, assume that ϕ has a satisfying assignment. For each i, we place a mine at x_i if x_i is true and at $\neg x_i$ otherwise. This is consistent with t_i's label. Next, for each C at least one among the three literal vertices adjacent to h_C contains a mine, since C is satisfied. Then, we can place some extra mines at either u_C or v_C or both to make h_C consistent.

   On the other hand, assume that G has a consistent placement of mines, then for each variable x_i, either x_i or $\neg x_i$ contains a mine but not both (because of t_i's label). We set x_i to T if it contains a mine and F otherwise. We claim that this is a satisfying assignment. Consider any clause C, since u_C and v_C provide at most 2 mines, at least one of the three literal vertices adjacent to h_C must contain a mine in order for h_C to be consistent. This means one of the three literals of C is set to T, hence satisfying C.

   Thus, MINES-CONSISTENCY is NP-complete.

4. **LDC is in** NP. A polynomial size certificate for LDC is the lists of objects in the k clusters. To verify this certificate, the verifier computes the distances between any two objects in the same cluster and check them against B. This verfication takes $O(n^2)$ time.

   LDC is NP**-hard**. We give a reduction from k-Colorability to LDC. Given a graph G as an instance of k-Colorability, we construct an instance of LDC as follows. For each vertices G, we create a corresponding objects - we will use the same names for the vertices and the objects. If there is no edges between two vertices u and v, we let $d(u, v) = B$. Otherwise, we let $d(u, v) = B + 1$.

   We claim that G can be colored by k colors iff the objects can be partitioned into k clusters. On one hand, assume that G can be colored by k colors. Then the objects corresponding to vertices of each color form a cluster, since the distance between any two of them is B.

   On the other hand, assume that the objects can be participated into k cluster. Consider any two objects u and v in the same cluster. Since the $d(u, v) \leq B$, there is no edge between u and v in G. This means we can color all the vertex corresponding to objects in each cluster by the same color; thereby coloring G by k colors.

   Thus, LDC is NP-complete.

5. **Idea:** Let's start by constructing the cluster that contains the first object p_1. First, all the objects p_i such that $d(p_1, p_i) < B$ has to be in this cluster. Then, all the objects that has distance at most B

to one of these p_i must also be in the cluster. This process continues until we don't need to add any more objects to the cluster. Then we can start to build the next cluster. Note that in this problem, the more clusters we can construct the better: if the answer is "yes" for k, it is also "yes" for $k-1$ (verify this).

Algorithm:

```
while there are unmarked objects:
  if there is an unmarked pi such that d(pi,pj) < B for some pj in the current cluster:
    add pi to the current cluster
    mark pi
  else:
    create a new cluster containing one unmarked object pi
    mark pi
if at least $k$ clusters were created:
  return yes
else:
  return no
```

Running time: The while loop takes n iterations. In each of iteration, testing the condition of the if statement take $O(n^2)$ time. Thus, the algorithm runs in $O(n^3)$ time.

Correctness Proof: The correctness of the algorithm follows from the fact that each cluster it constructs is minimal: all the points in such a cluster must stay together in order to guarantee that the inter-cluster distance is at least B.

Note: You can also solve this problem by reducing it to Minimum Spanning Tree as follow. First, construct a graph where each vertex represents an object and the length of the edge connecting vertex is the distance between the to represented objects. Next, compute the minimum spanning tree of the graph. If there are $k-1$ edges of length at least B in the spanning tree, then the answer is "yes", otherwise, the answer is "no".

First, assume that such $k-1$ edges exist; then removing them separate the spanning tree into k subtrees. The objects in each subtree form a cluster. The smallest distance between two objects in two clusters T_i and T_j is the length of the original edge connecting T_i and T_j; for if there is another edge whose distance is smaller, the minimum spanning tree should have used it.

On the other hand, assume that there are k clusters such that the minimum distance between any two clusters is at least B; then the spanning tree must contain at least $k-1$ edges among the inter-cluster edges. So it contains at least $k-1$ edges of length at least B.

Note 2: The reduction to minimum spanning tree is nicer than greedy algorithm above in the sense that given k, it finds the maximum minimum inter-cluster distance.