CS 344 Homework #4

Austin Bennett, Jeremy Barkalow

December 6, 2019

Question #1

Our goal is to show that if TSP can be solved in polynomial time, then so can TSP-OPT. We solve this by showing that TSP-OPT can be solved by making a polynomial number of calls to TSP. Since TSP can be solved in polynomial time the overall time complexity would be polynomial.

```
TSP-OPT(G):
    A = 0
    For all g ∈ V, h ∈ V:
    A = A + dist(g, h)
    B = A
    C = 0
    While(C < A)
    {
        budget b = (C + B)/2
        TSP(G, b)
        If(TSP returns tour)
        {
            B = b
            Tour = tour that TSP returned
        }
        Else
        {
            C = b
        }
    }
    Return Tour
```

Question #2

We can return a Hamiltonian Path in polynomial time using a procedure which itself runs in polynomial time that tells us if the graph has a hamiltonian path by making a polynomial number of calls to that procedure. This would be done by choosing each edge e from our graph and removing this edge and passing this modified graph into our procedure to test if that new graph without edge e still contains a hamiltonian path. If the procedure returns that this new graph does not contain a hamiltonian path. We add edge e back to our current graph permanently. If the procedure returns that the new graph still contains a hamiltonian path after the removal of edge e, we remove e permanently and update the current graph to not contain e and move onto the next edge until all the edges are tested. The edges left after the end of this procedure will be a hamiltonian path.

Question #3

To show that Π, a problem in NP, can be solved in $O(2^{p(n)})$ we reduce the problem Π to a SAT problem. Since any problem of size n can be reduced to a SAT problem with size p(n), coupled with the fact that a SAT problem with size p(n) can be solved in $2^{p(n)}$ time we have that any problem Π can also be solved in $2^{p(n)}$ time. Since we can generalize Π to a SAT problem it is clear that in the worst case we can solve any Π through this method.

Question #4

(a) Solvable in polynomial time. We start off by deleting all the vertices in the set L from our graph. If no spanning tree exists for this new graph (the graph is unconnected) then no spanning tree exists where all the vertices in L are leaves. If a spanning tree does exist, we should be able to add the deleted vertices back to the graph by attaching each vertex to any of its previous neighbors and as such this graph will have all vertices in L as leaves.

(b) This situation can be generalized to the undirected (s,t) Rudrata Path problem. Let us set L = {s, t}. Given a spanning tree of the graph G, and that the only leaves of the tree can be {s, t} we have that: a path must exist from s to t, and as such it must be a Rudrata path because a spanning tree must include all vertices of the graph. Since the situation can be generalized to an NP-complete problem then the situation is certainly NP-complete.

(c) We can use the same generalization as for (b). Since "its set of leaves included in the set L." is just a subset of "precisely the set L", we can again obtain the case where L is exactly {s, t} and the same proof follows.

(d) This situation can be generalized again to the Rudrata path problem. Given the case where the graph G has two vertices, both of the vertices are leaves. The graph itself is also the spanning tree and as such the path from s to t must be a path containing all vertices. Therefore this situation is NP-complete.

(e) For this situation we can use the reduction from vertex cover to a dominating set such that if we connect all vertices in the graph G to every other vertex in the graph we obtain a graph that is necessarily connected. A graph only has a connected dominating set of size at most k if and only if it has a spanning tree with $|V|$ - k or more leaves, however. Thus, our minimum complexity is that of dominating set which is NP-complete.

(f) Similarly to (c) and (b), situation (f) is a subset of situation (d) when k = 2, and the same generalization follows from (d). Therefore NP-complete.