

# CS 7910 Computational Complexity

## Assignment 6

---

Gopal Menon

March 5, 2016

1. **(20 points)** In this exercise, we consider the *Traveling Salesman Problem (TSP)*.

Let  $G$  be a **complete directed graph**. Each directed edge  $(v_i, v_j)$  has a weight  $w(v_i, v_j)$ . It is possible that  $w(v_i, v_j) \neq w(v_j, v_i)$ , i.e., the edge weights can be asymmetric.

The optimization version of the TSP problem is to find a cycle  $C$  in  $G$  containing each vertex of  $G$  exactly once such that the total weight of the edges of  $C$  is minimized.

The decision problem of TSP is as follows. Given a value  $W$ , decide whether  $G$  has a cycle  $C$  that contains each vertex of  $G$  exactly once such that the total weight of the edges of  $C$  is at most  $W$ .

Prove that the decision problem of TSP is NP-Complete (Hint: by a reduction from the Hamiltonian Cycle problem).

The Hamiltonian Cycle problem is known to be NP-Complete.

2. This is a “warm-up” exercise on designing approximation algorithms.

A variation of the *knapsack problem* is defined as follows (actually it is the subset-sum problem): Given as input a knapsack of size  $K$  and  $n$  items whose sizes are  $k_1, k_2, \dots, k_n$ , where  $K$  and  $k_1, k_2, \dots, k_n$  are all positive real numbers, find a full “packing” of the knapsack (i.e., choose a subset of the  $n$  items such that the total sum of the sizes of the items in the chosen subset is *exactly* equal to  $K$ ).

We have proven that this problem is NP-complete, which implies that it is very likely that efficient algorithms (i.e., polynomial-time algorithms) for this problem do not exist. Thus, people tend to look for good **approximation algorithms** for solving this problem. In this exercise, we relax the constraint of the knapsack problem as follows. We still seek a packing of the knapsack, but we need not look for a “full” packing of the knapsack; instead, we look for a packing of the knapsack (i.e., a subset of the  $n$  input items) such that the total sum of the sizes of the items in the chosen subset is at least  $\frac{K}{2}$  (but no more than  $K$ ). This is called a *factor of 2 approximation solution* for the knapsack problem. To simplify the problem, we assume that a factor of 2 approximation solution

for the input always exists, i.e, there always exists a subset of items whose total size is at least  $\frac{K}{2}$  and at most  $K$ .

- a) **(20 points)** Design a *polynomial-time* algorithm for computing a factor of 2 approximation solution, and analyze the running time of your algorithm (using the big- $O$  notation).
- b) **(5 points)** Improve your algorithm to  $O(n)$  time. If your algorithm for part (a) already runs in  $O(n)$  time, then you will get the five points automatically.

**Note:** You are required to clearly describe the main idea of your algorithm. Although the pseudo-code is not required, you may also give the pseudo-code if you feel it is helpful for you to explain your algorithm. You also need to briefly explain why your algorithm works, i.e., why your algorithm can produce a factor of 2 approximation solution. Finally, please analyze the running time of your algorithm.