Pre-tutorial questions

Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. What does it mean in practice that a problem is NP-hard or NP-complete?
2. What is a c-approximation algorithm?
3. If we can solve the Vertex Cover problem for trees in polynomial time why doesn't this contradict the fact that Vertex Cover problem is NP-complete?

Tutorial

Problem 1

Consider the following simple greedy algorithm that processes the items in arbitrary order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin. Prove that the greedy algorithm is a 2-approximation algorithm for the bin-packing problem.

Problem 2

Consider the following algorithm for the Euclidean travelling salesman problem. As input we are given a set P of n points in the plane. Compute a minimum spanning tree of P. Double all the edges such that a tour T is formed. Prove that T is at most twice a long as an optimal solution for the Euclidean travelling salesman problem.

Problem 3

As input, Knapsack takes a set of n items, each with value v_i and weight w_i, and a knapsack with weight bound W (for simplicity we assume that all elements have $w_i < W$). Find a subset I of the items that maximizes the value $\sum_{i \in I} v_i$ subject to the constraint $\sum_{i \in I} w_i \leq W$.

Knapsack is known to be NP-hard. Using dynamic programming, we can get an exact solution for knapsack in time $O(nW)$. Unfortunately, this is not polynomial in the size of its representation since W could be very large. Thus, we go back to our toolbox of approximation algorithms.

1. Lets try a basic greedy algorithm. Intuitively, we want the items with the most bang for the buck.

Algorithm 1 GreedyKnapsack

procedure GreedyKnapsack($\{v\}, \{w\}, W$)
    Sort items in non-increasing order of v_i/w_i.
    Greedily pick items in the above order as long as adding an item to the collection does not exceed the capacity of the knapsack.
end procedure

Prove that this algorithm can be arbitrarily bad.

2. We make the following small adjustment to our greedy algorithm:

Algorithm 2 GreedyKnapsack Redux

```
procedure GreedyKnapsack({v}, {w}, W)
    Sort items in non-increasing order of v_i/w_i.
    Greedily pick items in the above order until we hit an item i that
  is too big to fit.
    Select either {1, 2, ..., i − 1} or item {i}.
end procedure
```

At first glance, this seems to be a clumsy hack in order to deal with the above counterexample. However, it turns out that this algorithm has far better performance. Prove that this algorithm is a 2-approximation algorithm.

Problem 4

Consider the following problem defined on a set of clients C and a set of facilities F. Each facility $i \in F$ has associated an opening cost f_i and for each client-facility pair (j, i) there is a cost $d_{j,i}$ associated with client j being assigned to facility i. (You can think of $d_{j,i}$ as the distance from j to i.) The objective of the problem is to open a subset of the facilities so that clients can be assigned to a close-by facility.

Given F, C $f : F \to \mathbb{Z}^+$ and $d : C \times F \to \mathbb{Z}^+$, the *facility location* problem is to choose a set $X \subset F$ minimizing

$$\sum_{i \in X} f_i + \sum_{j \in C} \min_{i \in X} d_{j,i}.$$

1. Show that the facility location problem NP-hard

2. Give an $H_{|C|}$ approximation algorithm based on the greedy algorithm for set cover.

Problem 5

Consider the following local search algorithm for vertex cover with the objective of minimizing the size of the cover. The algorithm start with the complete set of vertices. Let S be a vertex cover. There are two types of local moves while at S:

1. Remove one vertex from S

2. Remove two vertices from S and add one to it.

The only restriction is to maintain feasibility. Show that this algorithm can get stuck in local optima that are much larger than the global optimum.