Fall 2021 CS: 4310 Homework 8

The *Traveling Salesman Problem (TSP)* is a well-known NP-complete problem. It takes as input a set of cities and pairwise distances between these cities and outputs a shortest *tour* that visits every city exactly once and returns to the starting city. More precisely, the input to the problem is an n-vertex complete graph $G = (V, E)$ in which each edge $e \in E$ has a positive weight $w(e)$. The output is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that

$$\sum_{i=1}^{n-1} w(v_i, v_{i+1}) + w(v_n, v_1)$$

is minimized.

Your task for this project is to implement 2 algorithms for the TSP problem and present the results of some experiments with these algorithms. For your experiments you will use the input file `miles.dat` that I have posted, containing pairwise distances among 128 cities in North America.

1. **MST-based 2-approximation algorithm:** When the weights of the edges satisfy the metric property, then it is possible to obtain a 2-approximation algorithm to TSP by using the miminimum spanning tree. This 2-approximation algorithm is described in these lecture notes `http://www.cs.tufts.edu/~cowen/advanced/2002/adv-lect3.pdf` by Professor Lenore Cowen from Tufts University. It is enough to read Section 1 and Section 2.1 in order to understand this simple algorithm and why it is a 2-approximation algorithm. Since it is essentially MST computation followed by DFS, it runs in $O(m \log n)$ time. Since the input to TSP is the complete graph, the running time can be rewritten as $O(n^2 \log n)$.

2. **Branch-and-bound algorithm:** The basic branch-and-bound algorithm for this problem is quite simple. Suppose we have somehow computed a tour T, representing our current best solution to the problem. We would now like to systematically generate all possible tours, rejecting long (i.e., bad) tours as quickly as possible. Specifically, as soon as we have a partial tour whose length is longer than the length of T, we should reject that tour immediately and move on to generating the next tour. On the other hand if we have completed the generation of a tour, say R, then that must mean that R is shorter than T, and therefore we replace T by R. When I say "partial tour" I mean a tour that starts at a city, visits a few cities (not necessarily, all) exactly once and returns to the starting city. Here is the pseudocode for a function called TSP that expresses this algorithmic idea.

```
// R is a partial tour; initially, it may be empty or just contain the first city
// T is the current best tour; it may be computed greedily, initially
// Invariant: length(R) < length(T)
Branch-and-BoundTSP(R, T)
        // Base case: we have discovered a tour better than T
        if (R is a complete tour)
                T <--  R;
                return;

        // Recursive case: R is not complete and is currently better than T
        // and is therefore worth completing
```

```
        for (each city c not in R) do
                append c to R;
                if (length(R) >= length(T)) then
                        delete the last city from R
                else
                        TSP(R, T);
        end of for-loop
end of Branch-and-BoundTSP
```

The use of the "current best" tour T to prune our search space is critical and is our only hope that we can compute a traveling saleman tour of 128 cities in a reasonable amount of time. Notice that as the algorithm proceeds, T gets shorter and this may lead to quicker pruning later. Initially, T can be computed greedily using the *nearest neighbor heuristic*: repeatedly pick the nearest neighbor (not in the tour) of the most recent vertex in the tour.

Details on experiments to perform and what to submit will be provided next week.