CSE6140/CX4140 Fall 2019 TSP Project (Problem description)

October 25, 2019

1 Overview

The Traveling Salesman Problem (TSP) arises in numerous applications such as vehicle routing, circuit board drilling, VLSI design, robot control, X-ray crystallography, machine scheduling and computational biology. In this project, you will attempt to solve the TSP using different algorithms, evaluating their theoretical and experimental complexities on both real and random datasets.

2 Objective

- Get hands-on experience solving an intractable problem that is of practical importance
- Implement an exact branch-and-bound algorithm
- Implement approximate algorithms that run in a reasonable time and provide high-quality solutions with concrete guarantees
- Implement heuristic algorithms (without approximation guarantees)
- Develop your ability to conduct empirical analysis of algorithm performance on datasets, and to understand the trade-offs between accuracy, speed, etc., across different algorithms
- Develop teamwork skills while working with other students

3 Groups

You will be in a group of up to 4 students. We encourage you to form your own groups, and use Piazza as a resource. Afterwards, please designate **one** member per group to complete the assignment "Group and programming language submission for project" through Canvas:

In the text entry, put each student's last name and first name per line, followed by the programming language (Python or C++) your group will use. For example:

```
Fox, James
An, Xiaojing
Chunxing, Yin
Chen, Liang
Python
```

4 Background

We define the TSP problem as follows: given the x-y coordinates of N points in the plane (i.e., vertices), and a cost function $c(u, v)$ defined for every pair of points (i.e., edge), find the shortest simple cycle that visits all N points. This version of the TSP problem is *metric*, i.e. all edge costs are symmetric and satisfy the triangle inequality.

5 Algorithms

You will implement 4 algorithms total, that fall into three category counts: 1 Exact, 1 Heuristics, 2 Local Search:

1. exact, computing an optimal solution to the problem
2. construction heuristics, some of which have approximation guarantees
3. local search with no guarantees but usually much closer to optimal than construction heuristics.

In what follows, we present the high-level idea behind the algorithms you will implement.

- **Exact algorithm using Branch-and-Bound**. Implement the Branch-and-Bound algorithm as seen in class. We present several approaches to compute the lower bound function (please use either the 2 shortest edges, or the MST bounding functions, or something stronger you find in the literature) in class. Feel free to read up on what researchers have proposed for this problem. You may design any lower bound of your choice as long as it is indeed a lower bound.

  Since this algorithm is still of worst-case exponential complexity, your implementation must have additional code, similar to all of your other implementations, that allows it to stop after running after some amount of time and to return the current best solution that has been found so far. Clearly, for small datasets, this algorithm will most likely return an optimal solution, whereas it will fail to do so for larger datasets.

- **Construction Heuristics with approximation guarantees**. Please choose and implement one construction heuristic.

  MST-APPROX 2-approximation algorithm based on MST detailed in lecture

  FARTHEST-INSERTION insert vertex whose minimum distance to a vertex on the cycle is maximum

  RANDOM-INSERTION randomly select a vertex and insert vertex at position that gives minimum increase of tour length

  CLOSEST-INSERTION insert vertex closest to a vertex in the tour

  NEAREST NEIGHBOR

  SAVINGS HEURISTIC

- **Local Search**. There are many variants of local search and you are free to select which one you want to implement. Please implement 2 types/variants of local search. They can be in different families of LS such as SA vs Genetic Algorithms vs Hill Climbing, or they can be in the same general family but should differ by the neighborhood they are using, or by the perturbation strategy, etc. They need to be different enough to observe qualitative differences in behavior. Here are some pointers:

- Neighborhood - 2-opt exchange
- Neighborhood - 3-opt exchange or more complex one
- Perturbation using 4 exchange
- Simulated Annealing
- Iterated Local Search
- First-improvement vs Best-Improvement

6 Data

You will run the algorithms you implement on some real-world datasets.

The datasets can be downloaded from Canvas as `DATA.zip`.

In all datasets, the N points represent specific locations in some city (e.g., Atlanta).

The first several lines include information about the dataset, including the data type (Euclidean).

For instance, the `Atlanta.tsp` file looks like this:

```
NAME: Atlanta
COMMENT: 20 locations in Atlanta
DIMENSION: 20
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 33665568.000000 -84411070.000000
2 33764940.000000 -84371819.000000
3 33770889.000000 -84358622.000000
...
```

Note that the node ID is a unique integer assigned to each vertex, the x and y coordinates may be real numbers, and the three values are separated by spaces.

To get distance between any two points, compute the Euclidean distance (L_2 norm), then round to the nearest integer. A double with 0.5 decimal value is rounded up.

6.1 Example solution

One optimal tour solution, **Cincinnati.tour**, is also provided as example.