CSE 6140 / CX 4140

Computational Science & Engineering Algorithms

Project - Traveling Salesman Problem

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 solve the TSP using different algorithms, evaluating their running times on a number of datasets.

2 Objective

- Get hands-on experience solving an intractable problem that is of practical importance
- Implement an exact algorithm and observe its inefficiency.
- Implement an approximate algorithm that runs in a reasonable time and provides high-quality solutions with concrete guarantees.
- Implement a heuristic algorithm (without approximation guarantees).
- Conduct empirical analysis of algorithm performance and understand the trade-offs between accuracy and speed.
- Develop teamwork skills while working with others.

3 Groups

You will be in a group of up to 3 students. Please see the group sign-up spreadsheet.

4 Background

We define the TSP problem as follows: given the x-y coordinates of N points in the plane, find the shortest simple cycle that visits all N points. We'll consider the N points to be vertices of a graph. Edge costs are the Euclidean distance between those points.

This version of the TSP problem is *metric*: all edge costs are symmetric and satisfy the triangle inequality.

5 Algorithms

You will implement algorithms that fall into three categories:

1. exact: a brute-force solution to the problem
2. approximate: an approximate solution with quality guarantees
3. local search: a heuristic algorithm with no guarantees, but that is effective in practice.

For item 1, implement a brute-force solution with a time cut-off. That is, after T seconds (e.g. $T = 300$ seconds), the algorithm should output the solution found so far, and exit.

For item 2, implement the 2-approximation algorithm based on MST that we discussed in class.

For item 3, there are many variants of local search. You can use one of the algorithms presented in class (Hill Climbing, Simulated Annealing), or you can find other approaches in the literature (evolutionary algorithms, neural networks).

6 Data

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

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
...
```

The first column is vertex ID. The second and third columns are x and y coordinates. For example, node 2 is located in coordinates (33764940, -84371819) of a 2D plane.

To find the edge costs, compute the Euclidean distances between the vertices, then round to nearest integer.

7 Code

All your code files should include a top comment that explains what the given file does. Your algorithms should be well-commented and self-explanatory.

You must create an executable from your code. Your executable must take as input i) the filename of a dataset, ii) the method to use (BF: brute force, Approx: approximation algorithm, LS: local search), iii) the cut-off time (in seconds), and iv) a random seed (only used for your local search algorithm).

If your executable is run with the same 4 input parameters, your code should produce the same output. The executable must be run as follows (notice that arguments use 1 dash rather than 2 dashes):

```
exec -inst <filename>
    -alg [BF | Approx | LS]
    -time <cutoff_in_seconds>
    [-seed <random_seed>]
```

Where "exec" is the name of your executable.

Any run of your executable with the three or four inputs (filename, cut-off time, method, and if applicable based on method, seed) must produce an output file in the current working directory:

- File name: $\langle instance \rangle\_\langle method \rangle\_\langle cutoff \rangle\_[\langle random\_seed \rangle].sol$.
  - Note that "random_seed" is only applicable when the method of choice is randomized (e.g., local search).
  - For BF, "random seed" can be omitted, for approximation algorithm, "cutoff" can be omitted
  - e.g. when $cutoff = 600$, $seed = 42$, $instance = Atlanta$ the 3 types of algorithms would generate the following files with names: atlanta_BF_600.sol, atlanta_Approx_42.sol, atlanta_LS_600_42.sol
- File format: (a) line 1: quality of best solution found (a floating point number) (b) line 2: list of vertex IDs of the TSP tour (comma-separated): $v_1, v_2, v_3, ..., v_n$

You should run all the algorithms you have implemented on all the instances we provide, and submit the output files generated by your executable. See Deliverables section below.