CS 3510 - Spring 2020

Course project (and competition!)

Design your own Traveling Salesman Problem algorithm and try to make it as fast as you can

DUE DATE: APRIL 17, 2020, 11:59pm

Document version 1.0 (released March 17)

You can work on this project individually or in groups of two students.

1. Functional description and requirements

You are asked to design and implement a **Traveling Salesman Problem (TSP)** algorithm. In this problem you are given a set of nodes and the pairwise distances (edge costs) between nodes and you have to compute a Hamiltonian tour of minimum total length (i.e., visit each node only once and return back to the starting point of the tour).

You can use **any algorithm you want.** It does not need to be one of the algorithmic approaches we have covered in class. For instance, you can explore other ideas such as Local Search, Branch-and-Bound, Genetic Algorithms, Evolutionary Computing, Simulated Annealing, Ant Colony Optimization, etc. We encourage you to explore instead of quickly adopting an existing idea you read somewhere.

Please do not make any additional assumptions about the structure of the problem. For instance, you cannot assume that the distances between nodes follow the triangle inequality.

Even though the general approach of your algorithm may not be yours, the code should be yours – please, do not copy code from any online or offline resource. That would be considered cheating!

You can use **libraries but only for basic functions**, such as reading the coordinates of a set of points and computing pairwise distances. You are not allowed to use any libraries that are related to the TSP problem. If you plan to use some other libraries, please check with the TAs first.

Your program should be able to handle input files with **up to 1000 nodes.**

Note that we will not provide you with the pairwise node distances. Instead, we will provide you with x-y node coordinates and you will need to pre-compute the Euclidean distance between every pair of nodes at the start. **We suggest that you round all distances to the nearest positive integer so that your code is not slowed down by floating-point arithmetic.**

Supported programming languages

We recommend that you use Python-3.
If you absolutely need to use another programming language please coordinate with the TAs first because one of the TAs will need to run your code on her/his laptop. So, you will need to find a TA that can support the programming language you want to use.

The "Maximum Acceptable Tour" (MAT) requirement

To exclude algorithms that are really inefficient or naïve (such as selecting a tour randomly), we will provide you with a smaller test problem (around 30 nodes) and with the maximum acceptable tour length for that problem. If your algorithm produces a longer tour within 180 seconds (running on your laptop), you should design a better algorithm.

2. Command-line interface

The command-line should be as follows:

- `Command line:` **`tsp-3510 <input-coordinates.txt> <output-tour.txt> <time>`**

`<input-coordinates.txt>`: the name of a plain-text file in the same directory that includes the node-ID (a distinct positive integer) and the x-y coordinates for every node (one row per node). We will give you an example of this file so that there are no doubts about the format.

`<output-tour.txt>`: the name of a plain-text file that your program should create in the same directory. The first line should be the cost of the TSP tour you have computed. The second line should include the sequence of node-IDs (separated by a single blank character). Remember that the tour should return to the starting node!

`<time>`: the maximum number of seconds that your program should run. You will need to schedule an alarm so that your code reports the TSP tour and exits at that point.

Example: python3 tsp-3510.py coords.txt tour.txt 300

3. Submission instructions

You will need to submit a single ZIP/tar archive for the entire project.

The archive should include the python3 code, a README file (see below), the output that your code generated on the smaller test problem we gave you (see “MAT requirement”), and a PDF document in which you describe your algorithm (ideally in pseudocode) and explain the rationale behind its design. This pdf document will probably need to be just a couple of pages long.

The README file must contain :

- Your name (or names for group projects), email addresses, date of submission
- Names and descriptions of all files submitted
- Instructions for compiling and running your programs (if applicable)
- Any known bugs or limitations of your program that we should know about

An example submission may look like as follows -

tsp-3510.zip

| -- **tsp-3510/**

| -- **tsp-3510.py**

| -- README.txt

| -- mat-output.txt

| -- algorithm.pdf

Only one member of each group needs to submit the actual code and documentation. The other group member can submit a simple text file in which they mention their partner’s name that has submitted the actual assignment.

4. Grading and Competition

You will get full credit as long as you submit working code (that is yours!), satisfy the MAT requirement, and provide the required PDF document.

You can gain some extra points however if your code is among the top-10 submissions. The TAs will identify the top-10 projects on a common test problem (a tour that involves few hundreds of cities), running all submissions on the same machine, and for the same duration (around 3 minutes).

The top-10 submissions will receive a grade-bonus of 20% (i.e., 120 points instead of 100).