CSE6140/CX4140 Fall 2018 Project - TSP

Assigned: October 25th, 2018

1 Overview

The Traveling Salesperson 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. Please respond to the Canvas quiz for group assignments and you should be assigned to a group shortly after the quiz closes.

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.

Note that the cost function $c(u, v)$ is defined as either the Euclidean or Geographic distance between points u and v (see TSPLIB documentation for details `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/DOC.PS`).

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

For more details about types of TSP, please refer to the lectures slides.

5 Algorithms

You will implement algorithms that fall into three categories:

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. The slides 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). 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 presented in slides
  - Neighborhood - 3-opt exchange or more complex one
  - Perturbation using 4 exchange discussed in class
  - Simulated Annealing
  - Iterated Local Search
  - First-improvement vs Best-Improvement

6 Data

You will run the algorithms you implement on some real and random 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 or geographical). The format is as used in TSPLIB.

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.

Handling input and computing distances are as follows.

6.1 GEO Format

For the GEO format (x-coordinate is latitude, y-coordinate is longitude):

- Converting x-y coordinates to radians: use same formula as in the TSPLIB FAQ ("Q: I get wrong distances for problems of type GEO."), i.e. take the floor of the x and y coordinates. *Do not round to nearest integer.*
- Computing geographical distance between points i and j: use the exact same formula in the TSPLIB FAQ for `dij`; this formula rounds the distance to the nearest integer.

6.2 EUC_2D Format

There is no need to do anything with the x-y coordinates in the input files. Compute Euclidean distance then round to nearest integer. Note that a double with 0.5 decimal value is rounded up (also in FAQ).

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. Use a README file to explain the overall structure of your code.

Your executable must take as input i) the filename of a dataset and ii) the cut-off time (in seconds) iii) the method to use, iv) a random seed. The arguments cannot be changed and if your submitted code does not compile to or result in an application that executes with these arguments, you will lose a *significant number* of points. If you have any questions, please contact the TAs well

in advance of the submission deadline. If it is run with the same 4 input parameters, your code should produce the same output. The executable must conform with the following arguments / execution:

```
exec -inst <filename>
        -alg [BnB | Approx | LS1 | LS2]
        -time <cutoff_in_seconds>
        [-seed <random_seed>]
```

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 two types of output files in the current working directory:

1. Solution files:
   - File name: `<instance>_<method>_<cutoff>[_<random_seed>].sol`, e.g. `instance_BnB_600.sol` or `jazz_LS1_600_4.sol`.

     Note that as in the first example above, `random_seed` is only applicable when the method of choice is randomized (e.g., local search). When the method is deterministic (e.g., branch-and-bound), `random_seed` is omitted from the solution file's name.
   - File format:
     (a) line 1: quality of best solution found (integer formatted in ASCII)
     (b) line 2: list of vertex IDs of the TSP tour (comma-separated, in ASCII): $v_1, v_2, v_3, ..., v_n$
2. Solution trace files:
   - File name: `<instance>_<method>_<cutoff>[_<random_seed>].trace`, e.g. `instance_BnB_600.trace` or `jazz_LS1_600_4.trace`. Note that `random_seed` is used as in the solution files.
   - File format: each line has two values (comma-separated):
     (a) A timestamp in seconds (double formatted in ASCII)
     (b) Quality of the best found solution at that point in time (integer formatted in ASCII). Note that to produce these lines, you should record every time a new improved solution is found.
     Example:

     ```
     3.45, 102
     7.94, 95
     ```

8 Output

You should run all the algorithms you have implemented on all the instances we provide, and submit the output files generated by your executable, as explained in the Code section.

Note: Save all of our resources (bits, electricity, time spent downloading your files) and do not submit any data files. Any submitted data files will be ignored.

9 Evaluation

We now describe how you will use the outputs produced by your code in order to evaluate the performance of the algorithms.

1. Comprehensive Table: include a table with columns for each of your TSP algorithms as seen below. For all algorithms report the time, your algorithms solution quality, and relative error with respect to the optimum solution quality provided to you in the TSP instance files. Relative error ($RelErr$) is computed as $(Alg - OPT)/OPT$). Round time and $RelErr$ to two and four significant digits beyond the decimal, respectively. For local search algorithms, your results for each cell should be the average of some number (at least 10) of runs with different random seeds for that dataset. You will fill in average time (seconds) and average solution quality. You can also report in any other information you feel is interesting.

	Branch and Bound			Etc. (other algorithms)		
Dataset	Time (s)	Sol.Qual.	RelErr	Time (s)	Sol.Qual.	RelErr
instance	3.26	3400	0.0021	...	....	...

The next three evaluation plots are applicable to local search algorithms only. All the information you need to produce these plots is in your solution trace files.

1. Qualified Runtime for various solution qualities (QRTDs): A plot similar to those in Lecture on Empirical Evaluation. The x-axis is the run-time in seconds, and the y-axis is the fraction of your algorithm runs that have 'solved' the problem. Note that 'solve' here is w.r.t. to some relative solution quality q^*. For instance, for $q^* = 0.8\%$, a point on this plot with x value 5 seconds and y value 0.6 means that in 60% of your runs of this algorithm, you were able to obtain a solution quality at most the optimal size plus 0.8% of that. When you vary q^* for a few values, you obtain the points similar to those presented in class.

2. Solution Quality Distributions for various run-times (SQDs): Instead of fixing the relative solution quality and varying the time, you now fix the time and vary the solution quality. The details are analogous to those of QRTDs.

3. Box plots for running times: Since your local search algorithms are randomized, there will be some variation in their running times. You will use box plots, as described in the 'Theory' section of this blog post: `http://informationandvisualization.de/blog/box-plot`. Read the blog post carefully and understand the purpose of this type of plots. You can use online box plot generators such as `http://shiny.chemgrid.org/boxplotr/` to produce the plot automatically from your data.

10 Report

1. Formatting

   You will use the format of the Association for Computing Machinery (ACM) Proceedings to write your report. Please use the templates from `https://www.acm.org/publications/proceedings-template`.

2. Content

   Your report should be written as if it were a research paper in submission to a conference or journal. A sample report outline looks like this:

   - Introduction: a short summary of the problem, the approach and the results you have obtained.
   - Problem definition: a formal definition of the problem.
   - Related work: a short survey of existing work on the same problem, and important results in theory and practice.
   - Algorithms: a detailed description of each algorithm you have implemented, with pseudo-code, approximation guarantee (if any), time and space complexities, etc. What are the potential strengths and weaknesses of each type of approach? Did you use any kind of automated tuning or configuration for your local search? Why and how you chose your local search approaches and their components? Please cite any sources of information that you used to inform your algorithm design.
   - Empirical evaluation: a detailed description of your platform (CPU, RAM, language, compiler, etc.), experimental procedure, evaluation criteria and obtained results (plots, tables, etc.). What is the lower bound on the optimal solution quality that you can drive from the results of your approximation algorithm and how far is it from the true optimum? How about from your branch-and-bound?
   - Discussion: a comparative analysis of how different algorithms perform with respect to your evaluation criteria, or expected time complexity, etc.
   - Conclusion

11 Deliverables

Failure to abide by the file naming and folder structure as detailed here will result in penalties. Failure for your code to run as clearly required above will result in penalties. If your code does not run as intended the correctness of your entire project can be in question and submitting code that the TAs can run correctly is a hard requirement.

1. Each group must submit a Report: a PDF file of the report following the guidelines in section Report.

2. Each student should submit the same zip file.

   The zip file must have the following files/folders:

   - Code: a folder named 'code' that contains all your code, the executable and a README file, as explained in section Code.
   - Output: a folder named 'output' that contains all output files, as explained in section Code.

3. Each student should submit an evaluation of the team. For each team member (including yourself) include a score from 0 to 10, outline the contributions that the member did to the project, and justification for the score.

4. If your group wishes, it may enter a competition for bonus points. More details will be released on this later, and the first entry will be required two weeks prior to the project deadline.