Project – V
Transportation & Logistics

CSCI 2951-O: Foundations of Prescriptive Analytics

Due Date: December 11th, 2017 10PM

1 Problem Statement

The decision has been made! Based on the cost estimation found for the supply chain management business, the multi-billion dollar client of the Strategic Consulting Company (SCG) made a bold move and decided to enter the market. The facilities are open now, ready to ship their initial products while the customers are eagerly waiting for their orders to arrive. Rental arrangements are finalized, a fleet of vehicles is formed to serve the customers in various locations. The initial number of orders has exceeded the expectations and the business is booming!

In the next phase, transitioning from *strategic decisions* to the *operational side* of the business, what is missing is an efficient vehicle dispatching strategy to deliver products to customers. This a well-known transportation problem faced in the logistics industry. Everyday, delivery companies have to solve this problem routinely. One last time, SCG turns to CSCI 2951-O elites!

2 Vehicle Routing

Given a set of vehicles, the goal of this project is to design a route for each vehicle so that all of the customers are served by one vehicle and the travel distance of the vehicles is minimized. Additional complexity comes from the fact that the vehicles have a fixed storage capacity and the customers have different demands.

In this problem, we are given a list of locations $N = 0 \dots n-1$. By convention, location 0 is the warehouse (depot) location, where all of the vehicles start and end their routes. The remaining locations are customers. Each location is characterized by three values (d_i, x_i, y_i) $i \in N$ where d_i is the demand, and x_i, y_i denotes a geographical point. The fleet of vehicles $V = 0 \dots v-1$ is fixed and each vehicle has a limited capacity c. All of the demands assigned to a vehicle cannot exceed its capacity c. For each vehicle $i \in V$, let T_i be the sequence of customer deliveries made by that vehicle and let $dist$ be the Euclidean distance between two customers, calculated as $dist(j,k) = \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}$.

Then the delivery problem can be formalized as the following optimization problem:

$$
\begin{aligned}
\textit{minimize} \quad & \sum_{i \in V} \left(dist(0, T_{i,0}) + \sum_{\langle j,k \rangle \in T_i} dist(j,k) + dist(T_{i,|T_i|-1}, 0) \right) \\
& \sum_{j \in T_i} d_j \leq c \qquad \forall i \in V \\
& \sum_{i \in V} (j \in T_i) = 1 \qquad \forall j \in N \setminus 0
\end{aligned}
$$

3 Your Task

Your task is to build an algorithm to solve this optimization problem. You can use any complete or incomplete algorithm. This includes all the meta-heuristics studied in the lecture, and/or any previous solvers/algorithms you have seen so far.

As with your algorithm, you are also free to choose your favorite programming language to complete the project. You are encouraged to experiment with different strategies to attain the best results. You will share your results in class during team presentations. Keep in mind that having the results alone is not enough! You need to be able communicate our solutions clearly and in a compelling way. In this project, you will be evaluated not only based on your algorithm/results but also how you deliver your solutions.

You can assume that a *reasonable* time limit will be used to test the instances. Reasonable time is defined as the time it takes to leave your desk and get a fresh cup of coffee, which is roughly 5 minutes of running time for each instance on the department machines.

4 Support Code

We provide you with a data parser that reads instances and returns a simple instance object. This way, you can immediately start attacking the heart of the problem. The support code is written in Java.

You are free to use the support code as is, tweak the data structures as you see fit, or discard it completely. If your choice of programming language is different, the support code can serve you as a reference.

Assuming you are in the `project` directory,

To compile the support code, type:

```
> ./compile.sh
```

To run the support code in a given instance, type:

```
> ./run.sh input/5_4_10.vrp
```

In our running example, the expected output of the support code is:

```
> ./run.sh input/5_4_10.vrp
Number of customers: 5
Number of vehicles: 4
Vehicle capacity: 10
Depot location: 0 0
Instance: 5_4_10.vrp Time: 0.02 Result: N/A Solution: N/A
```

A feasible solution for this instance, let's call it 5_4_10_sol.vrp, is as follows:

```
80.6 0
0 1 2 3 0
0 4 0
0 0
0 0
```

The objective value is 80.6 followed by 0 meaning that the solution is *not* proved to be optimal (it would be followed by 1, if the solution is proved to be optimal). The route of each vehicle is given next. The first vehicle leaves the depot and visits the first three customers and then returns the depot. The second vehicle leaves the depot, visits the last customer and comes back to the depot. The remaining two vehicles is not assigned to any routes.

To help you analyze and improve your solutions, we also provide you with a solution visualizer. The visualizer can be found in the course page. It is a web application with a drag&drop interface that can visualize the routes found. It works by first providing the input data (e.g. 5_4_10.vrp), followed by providing the solution data (e.g, 5_4_10_sol.vrp).

5 Output

Given an input file as its first command line argument via the `run` script, your solution should print out the result on its **last line**. You can print other statements before the result but they will be discarded. We suggest you to keep such debug information to a minimum since writing I/O is expensive. We use `stdout` for output and ignore other streams. Your submission will be tested on department's linux machines and it should work without any other software package installation.

The expected output is "Instance:" prefix followed by the instance name, "Time:" prefix followed by the number of seconds it takes your program to solve this instance given in 2-digit precision, "Result:" prefix followed by the objective value found and "Solution:" prefix followed by the solution format used above printed on one line omitting the objective value.

```
./run.sh input/5_4_10.vrp
..running..bla..
..bla..
Instance: 5_4_10.vrp Time: 1.23 Result: 80.6 Solution: 0 0 1 2 3 0 0 4 0 0 0 0 0
```

Finally, to run a benchmark on all instances in a given input folder using a fixed time limit while collecting the result (i.e. the last line) to a log file, type:

```
> ./runAll.sh input/ 300 results.log
```

Please make sure to follow this convention *exactly*! Failing to do so will cause evaluating your submission incorrectly.

6 Submission

```
/solution
|_ src/
|_ compile.sh
|_ run.sh
|_ runAll.sh
|_ results.log
|_ presentation.pdf  <-- Notice: No report but a presentation!
|_ team.txt
```

- Submit all your source code under the (`/src`) folder. Remember that commenting your code is a good practice so that one can follow the flow of your program at a high-level.
- Submit your (possible updated) `compile` and `run` scripts which compiles your solution and runs it on a given instance respectively. Make sure that the *last line* of your solution output follows the specification above.
- Submit the last line found for all the instance in the input directory in the `results.log`.
- Submit your **presentation** (5-6 slides max) in pdf format which contains a discussion of your solution strategy, what worked and what did not, implementation techniques, and experimental observations. Pay attention to your presentation quality and clarity. Please also include a note on how much time you spend on this project. This information is completely irrelevant to evaluation and kept solely for statistical purposes.
- Submit the cs login(s) of the team in `team.txt` listing one member per line in lowercase.

Have questions? and/or need clarifications? Please do not hesitate to contact us, your instructor and TAs are here to help. Good luck on your quest for helping SCG!