Large Scale Optimization for Transport & Mobility

Assignment 1

Deadline: Friday September 19, 17:00

Rolf van Lieshout

1. This is an individual assignment. Therefore, it is **not** allowed to collaborate with others (helping each other with small coding questions is allowed). Copying code or text from others is considered plagiarism. If you use online sources (such as ChatGPT, Stack Overflow, but also Python libraries), indicate this clearly in the comments of your code.

2. Test your implementations on small instances you constructed yourself, and verify that the results are correct. It is often good practice to write a little bit of verification code to verify the correctness of your solutions.

GRASPing Outlier Insertion

In this assignment, you will develop and implement a Greedy Randomized Adaptive Search Procedure (GRASP) based on *outlier insertion* to solve the Traveling Salesman Problem (TSP). After selecting an initial city, the algorithm creates a tour by joining it with the city with the largest distance from the initial city. Then, in each iteration, among all cities not in the tour, we choose the city the furthest to any city in the tour. Finally, we insert the selected city in the position that causes the smallest increase in tour length. Figure 1 illustrates this outlier insertion heuristic for a small TSP instance.

To get you started, we have prepared code for the structure and basic functionalities of the program, which you can download via Canvas. The *zip-file* contains the following files:

- **`TSP.py`:** the Python file that contains a class that represents a Traveling Salesman Problem. The class already has methods for reading TSP instances, evaluating solutions and applying the nearest neighbor heuristic. You will have to extend this class.

- The folder "Instances": a folder with a large set of benchmark instances of different sizes.

You are asked to perform the following steps:

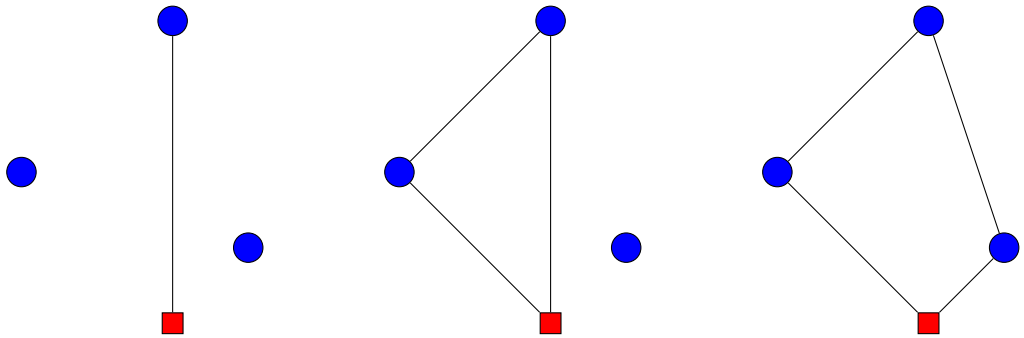

Figure 1: An example of running the outlier insertion on a small TSP instance

1. Make a selection of ten instances that you will use to test your code: five small instances, three medium instances and two large instances. Set a random seed to ensure that your results are reproducible.

2. Run the nearest neighbour heuristic for different starting positions. Comment on the variation in solution quality. (1 point)

3. Write a method `getTour_OutlierInsertion` that executes the outlier insertion as described above. (1 point)

4. There are two steps in the outlier insertion that can be randomized. Identify these two steps, and develop a GRASP algorithm by randomizing one of them, or both. Implement your algorithm in a method within the TSP class called `getTour_GRASPedInsertion`. (1 point)

5. Write a method in the TSP class called `isTwoOpt` that takes a tour as input and checks whether it is 2-optimal. Next, write a method called `makeTwoOpt` that takes in a tour as input and applies 2-exchanges until the tour is 2-optimal (i.e. `isTwoOpt` returns `True`). (1 point)

6. Develop an algorithm that combines the GRASPed insertion heuristic with 2-opt and implement this algorithm in the TSP class. Make a figure similar to the one on Slide 26 of the lecture slides on Local Search, with the costs before local search on the x-axis, and after local search on the y-axis. (1 point)

7. You are encouraged to compare different variations of the algorithms, test different insertion heuristics, enhance the efficiency and to perform sensitivity analyses. Creativity will be rewarded! (1 point)

Submission Requirements

Summarize your findings in a report of at most 6 pages (font size = 11pt, margin = 1 inch). Your report should include

1. a short introduction that states and motivates the problem and research (i.e., explains its relevance), summarizes the solution approach and the main results, ($\frac{1}{2}$ point)

2. a methodology section that outlines the developed algorithms, motivates the design choices (e.g. efficiency considerations or how you select the pair of arcs to be swapped in a 2-opt iteration), and discusses how you validated the correctness of your implementation, ($1\frac{1}{2}$ points)

3. a results section that presents and discusses the results of the computational experiments that you conducted, ($1\frac{1}{2}$ points)

4. a short conclusion that summarizes your research and discusses its limitations and how your algorithms/implementation could be improved, ($\frac{1}{2}$ point)

5. and an appendix that contains your Python code (the appendix does not count towards the 6-page limit).

Your submission should contain both the report in pdf format, and a zip-file that contains your code and the ten instances that you used to test your algorithms.

Assessment

This assignment will be assessed based on solution correctness and efficiency (0-6 points) and on the quality of the report (0-4 points). The sum of the obtained points directly translates to your grade.

In case anything about the assignment is unclear, please use the designated discussion board on Canvas.