Project I: TSP and Local Search

1 Introduction

This document describes the first project for the course, in which you will apply local search methods to the traveling salesperson problem. Make sure to **read all of the information** present in this document.

The goal of this project is to implement local search algorithms in a way that is optimized for a specific problem. Therefore, you are expected to implement not only the basic algorithms, but also different improvements you will design. Through experiments and analysis of results, you should then take conclusions on what is the best approach for this problem and data.

You will work in **groups of 3 students**, to implement algorithms, run experiments, and analyze the results. The project deliverables include a **comprehensive report** detailing your work, as well as all of the **code and data** that you generated. The evaluation will be based on the **quality of the approaches and analysis of results**, and will be judged mainly according to the report. The individual contribution of each member of the group will be tested through an **individual written assessment**.

The tentative deadline is **March 25**.

This document contains information about the project and what is expected from you, including:

Section 2: information about the traveling salesperson problem and the data to be used;

Section 3: advice on the methodology and how you should conduct your work, including some ideas to get you started;

Section 4: how to structure the report and what to submit at the end of the project;

Section 5: how the work will be evaluated.

2 The Traveling Salesperson Problem

In the traveling salesperson problem (TSP), we are given a graph $G = (V, E)$ with edge lengths $\ell \colon E \to \mathbb{R}_{\geq 0}$, and we want to find a cycle that visits every vertex and with minimum total length. We make two assumptions about the graph:

- it is complete, i.e. there is an edge between every pair of vertices,
- the edge lengths satisfy the triangle inequality

$$\ell(uv) \leq \ell(uw) + \ell(wv) \quad \text{for every } u, v, w \in V$$

The traveling salesperson problem is a great problem for research, because it is simple, useful in practice, but also hard to solve. It is thus not surprising that several new algorithmic techniques first appeared in the context of TSP, and that it continues to be one of the main benchmark problems when trying out new techniques.

2.1 Datasets

For the experiments, we will use a collection of instances (some generated, some obtained from real-world data), which you can obtain from the course page. The instances we consider are in the 2D plane, and use the Euclidean distance as the lengths of the edges. All of the instances come from three different sources: the TSPLIB project [3, 4], the DIMACS challenge [2], and some hard instances by Hougardy and Zhong [1].

You are encouraged to try to generate your own random instances using a method of your choosing, and to include the results in the report. Larger instances are available from the DIMACS challenge, and were not included to not make the files unnecessarily large. If you would like to use some of those, let me know and I will provide some.

2.2 TSPLIB file format

The provided instances are in the TSPLIB format, which looks like the example below:

```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin (Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
...
```

The first few lines of the file contain a name (`NAME`), a small description (`COMMENT`) and number of vertices (`DIMENSION`). They also contain a field `TYPE` which is always `TSP`, and a field (`EDGE_WEIGHT_TYPE`) which specifies how to compute the edge weights. There can be any number of spaces before and after the colon (`:`) on each line.

The next part of the file contains information about the vertices, starting with `NODE_COORD_SECTION`, and then one line for each vertex, containing the integer index of a vertex followed by its x and y coordinates in the plane (e.g. vertex 1 has coordinates $(565.0, 575.0)$).

The graph is always complete, and the edge weight of an edge (v_i, v_j) is always obtained by rounding the Euclidean distance,

$$d\big((x_i, y_i), (x_j, y_j)\big) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

If `EDGE_WEIGHT_TYPE` has the value `EUC_2D`, the distance is rounded to the nearest integer, and if it is `CEIL_2D`, it is rounded to the next integer. While the rounding is not particularly important for the project, it can be helpful to compare with the known results for those instances.

3 Methodology

The objective of this work is to implement and compare local search algorithms to find approximate solutions to TSP. You must implement greedy local search and simulated annealing in their basic forms, using the vertex-switching and 2-opt operators. You should also implement at least one more neighborhood operator.

As to the methods used, you are encouraged to implement a different selection strategy for greedy local search (besides the default strategy), and should experiment with different schedules for simulated annealing.

Though there are many advanced techniques for this problem, we will focus on greedy local search and simulated annealing, and will purposefully exclude more advanced techniques such as the Lin-Kernighan heuristic and other state-of-the-art techniques.

Your implementation should allow for easy selection of different initialization strategies, neighborhood structures, and methods. For that purpose, we recommend the library argparse, though you are free to use whatever method you prefer.

3.1 Experiments

The experiments serve to fairly evaluate the algorithms and different variations. Thus, there are several aspects to consider so that you can safely take conclusions from the results.

There are two measures that are of interest to us: running time and solution fitness. For the running time, it is important to run the experiments in the same machine, and under similar conditions; the characteristics of the machine should be specified in the report. As to the solution fitness, it is important to not take an average of values of different instances, which may have incomparable optimum values. Instead, you can calculate a ratio to a reference algorithm (e.g. basic local search) and compute the averages of that measure. You should only do this if the instances have the same size and are similar or generated using the same method, such as in the DIMACS data set. It remains however important to analyze instances individually.

Each data point should be an average of at least 30 experiments, especially since we are working with randomized algorithms. In particular, this means that when evaluating an algorithm on a single instance, you should execute it 30 times, and report the best, average, and standard deviation. If you are evaluating groups of instances (e.g. from the DIMACS project), you can use a combination of number of instances and repetitions such that the total is above 30 (e.g. 6 instances with 5 repetitions each).

Finally, each experiment should focus on the impact of a single parameter, and it must be clearly described in the report, including information about which parameter changes and how. It is recommended that you use experiments to help you pick the best parameters for each algorithm, before doing a final evaluation comparing the best versions of each approach.

Remember that the goal of the experiments is to have a **fair comparison** of the algorithms, to allow you to interpret results and obtain solid conclusions. While there are more rigorous statistical methods of algorithm evaluation, these are out of scope for the course, and thus we use the simpler analysis described in this section.

3.2 Some Tips and Ideas

- Test various neighborhood operators to see their impact on solution quality and convergence speed;
- Tune algorithm parameters (e.g., temperature schedule for simulated annealing) through empirical testing;
- Visualize TSP solutions using plots to compare routes found by different algorithms, illustrate the algorithms in the report, and to help you find bugs;
- Experiment with different ways to generate an initial tour and how to select the neighbor in each iteration.

4 Report and Deliverables

The project submission consists of two parts:

1. A written report, fully describing your approach and the results you obtained;
2. Your experimental setup, including the code you wrote, the raw data of the results, a sample of the instances you generated and of the solutions you computed.

4.1 Report

For the report, you are expected to write a structured document, around 10 pages, describing the entirety of your approach. It should contain the following sections:

- Introduction: a high-level overview of the approaches, methodology and results.
- Methodology: describing in detail the approaches used, how they were modified for the problem, and everything that brings value to your work;
- Results and discussion: the data in summarized form (plots, statistical measures), as well the interpretation of the data (what is surprising, what was expected, any interesting observations);
- Conclusion: A summary of what were the goals, what you achieved, and the main conclusions of the work.

Formatting guidelines: Use headings and subheadings to organize information logically; ensure consistent formatting, such as font size, spacing, and margins; and incorporate visual elements like charts, tables, or bullet points to enhance readability.

Before submitting, make sure to proofread the report carefully to maintain clarity, coherence, and accuracy throughout.

4.2 Deliverables

At the end of project, you must submit two files: the report in pdf format, and a zip archive containing your code and all other useful data. These files must be submitted on the page of the course before the deadline.

Your code should be well-documented (using comments) and all of the required files must be present in the zip archive, along with instructions on how to execute it (in a file named `README`). If you generated any data, you should include it in the zip archive or specify how to generate it. You must also provide all of the raw data that is used in the report.

5 Evaluation

The projects will be evaluated based on the following criteria:

- Implementation of the base algorithms;
- Level of optimization and improvement over the base approaches;
- Description of the implementation and improvements in the report;
- Experimental setting;

- Analysis and interpretation of the results;

And to a lesser extent:

- Writing quality and structure of the report;
- Code quality and documentation.

The overall grade of the group is multiplied by an individual factor, which will be determined by a written evaluation. Each member of the group must be capable of explaining any part of the code, discuss the experiments that were performed, and be aware of the content of the report.

References

[1] Stefan Hougardy and Xianghui Zhong. The $T_{n,m}$ instances in TSPLIB format. Available at https://www.or.uni-bonn.de/~hougardy/HardTSPInstances.html, 2018.

[2] David Johnson, Lyle McGeoch, Fred Glover, and Cesar Rego. 8th DIMACS implementation challenge: The traveling salesman problem. Available at https://dimacs.rutgers.edu/archive/Challenges/TSP/, 2000.

[3] Gerhard Reinelt. TSPLIB – A traveling salesman problem library. *INFORMS J. Comput.*, 3(4):376–384, 1991.

[4] Gerhard Reinelt. TSPLIB – library of traveling salesman problems. Available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, 1995.