U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Analysis and Synthesis of Algorithms
Design of Algorithms (DA)

Spring 2024
L.EIC016

Programming Project II

Routing Algorithm for Ocean Shipping and Urban Deliveries

DA 2024 Instructors Team

Departamento de Engenharia Informática (DEI)/Departamento de Ciências de Computadores (DCC)
Faculdade de Eng. da Univ. do Porto (FEUP)/Faculdade de Ciências da Univ. do Porto (FCUP)

Spring 2024

Due Date: May 18, 2024 at midnight (PT time)

1. Objectives

We have seen during the theoretical classes that some problems are *intractable*, that is, there is no efficient solution to solve them. This category of problems is not only academic, as they often show up in realistic real-world applications that require some kind of solution. In the second programming assignment, you will analyse the **Travelling Salesperson Problem (TSP)** and design heuristics to solve it, using several datasets from the context of ocean shipping and urban deliveries.

The goal of this assignment is twofold. First, implement a basic exhaustive approach for the classic routing problem using the TSP abstraction, therefore learning first-hand that although such an approach can find optimal solutions, its applicability is restricted to very small graphs. Second, refine your **critical thinking** skills, by developing and analysing a set of approximate solutions to the TSP. You will work in groups of 2 or 3 students (3 is preferable) to encourage interpersonal and project management skills. The deliverables for this project are both a description of an algorithm and its implementation alongside a short presentation.

We next describe the project's motivation, a short overview of the dataset you will work with, the problem statement and the instructions to prepare a presentation and demonstration of your work. The problem statement includes the description of each task (with the corresponding grading). Lastly, we provide specific turn-in instructions you need to follow. **Recall, the deadline is May 18, 2024 at midnight.**

2. Problem Statement

In this programming project, you are asked to design efficient algorithms to find optimal routes for vehicles in generic shipping and delivery scenarios, from urban deliveries to ocean shipping. This problem corresponds to the TSP. As the TSP is intractable, there are no known efficient algorithms to solve it. Backtracking techniques can find the optimal solution to this problem. If the graph is small, the application of these approaches might be reasonable. However, as you will conclude in this project, their application to large graphs is infeasible, due to their high computational complexity. Therefore, approximation algorithms based on heuristics need to be devised to efficiently address TSP-based problems. This class of approaches provide in many cases near-optimal solutions and may also prove bounds on the ratio between the approximate and the optimal solutions.

In this project you are asked to design efficient solutions to the TSP applied to diverse scenarios. You will be given multiple graphs and a starting point. While you are given plenty of freedom to explore effective heuristics, you are strongly suggested to at least implement the heuristic that takes advantage of a triangular approximation as this one has a guaranteed approximation bound for an important class of graphs. In particular, you should design appropriate heuristics according to the input data, such as the size and type of the graph, as well as additional information contained in the dataset. In this regard, you should analyse which heuristics are more adequate to a given graph and assignment and analyse the trade-offs between optimality and efficiency.

Additionally, you will implement a backtracking algorithm to the TSP. Although this algorithm is not efficient, it will enable you to determine the optimal solution to this problem for small graphs. These results can help you analyse the optimality of your heuristics on small graphs.

3. Problem Data and Basic Interface

To make your problem realistic, you are given several real datasets describing points of delivery in an urban setting as well as harbours in the context of ocean shipping. In addition, the provided data file also contains a set of very small graphs designed to allow you to visually test your algorithmic solutions. These large datasets are derived from realistic settings and are available on Moodle,

To adequately interface with your program, you ought to carry out the following three tasks:

[T1.1: 1.0 point] Create a simple interface menu exposing all the implemented functionalities in a user-friendly interface. This menu will be instrumental to showcase the work you have developed in a short demo to be held at the end of the project.

[T1.2: 1.0 point] Develop basic functionality to load and parse the provided dataset files. This functionality, accessible through the menu, enables the selection of the dataset to be used in the analysis of the algorithms developed. With the extracted information, you must create one (or more) appropriate graphs upon which you will carry out the requested tasks. The modelling of the graph is entirely up to you, as long as it is a sensible representation of the data and enables the correct application of the required algorithms.

[T1.3: 1.0 points] Include documentation for all code implemented in this project, using Doxygen, indicating the corresponding time complexity for each algorithm implemented.

4. Statement of Work

To facilitate your work, we have structured the functionalities you are expected to develop into four sets as follows.

4.1. Backtracking Algorithm [T2.1: 4 points]

In this approach, you are asked to develop a backtracking algorithmic approach to the TSP for a graph starting and ending the node tour on node labelled with the zero-identifier label. You are expected to use the small graphs to validate this approach and to observe that for the larger graphs this approach is not really feasible. To this extent, you are suggested to plot the execution time (or other performance metrics you find significant) to illustrate the feasibility or not of this approach for larger graphs.

4.2. Triangular Approximation Heuristic [T2.2: 4 points]

In this approach, you are asked to use the geographic node data, and implement the approximation algorithm for TSP that relies on the triangular inequality to ensure a 2-approximation algorithm for this problem again starting and ending the node tour on node with the zero-identifier label. You should use the results from this algorithm and compare them with the backtracking algorithm for the same small graphs.

If you load the real-world graphs, you will notice that they are actually not fully connected. However, to ensure the 2-approximation to the optimal result, you should **assume that all given graphs are fully connected**. As such, and only whenever you need to determine the distance between two nodes that is not provided, you need to use the geographic latitude and longitude distance to determine "on demand" the distance between two nodes and assume they are therefore connected. This is in essence equivalent to having a fully connected graph but without explicitly representing the edges. In order to compute the distances between nodes, you need to use the Haversine distance computation described in appendix A.

4.3. Other Heuristics [T2.3: 4 points]

In this approach, you are asked to implement another heuristic of your choice to solve the TSP again starting and ending the node tour on node with the zero-identifier label. The emphasis in this approach should be on efficiency as this algorithm should be feasible even for the large graphs. Here you can make use of several algorithmic techniques from the use of divide-and-conquer to splitting the graph into multiple geographic sections to the use of clustering, or both, possibly even in combination with the 2-approximation algorithm. In the end, you are strongly suggested to compare the quality and run-time performance of your heuristic solution against the 2-approximation algorithm. You are expected to present only **one** heuristic that works well over the basic Triangular approximation or one that offers comparable tour length results but with noticeably faster execution time (i.e., time complexity). You should avoid investing a tremendous effort in developing an heuristic that yields marginal gains. It is also important to demonstrate what gains you achieved with your heuristic, so, during the presentation, a comparative analysis between the backtracking, the triangular approximation and your algorithm should be shown.

Once again, in order to directly compare your results with the triangular approximation heuristic, you should assume that all given graphs **are fully connected** and use the Haversine distance to compute the weights of the edges that are not explicitly described in the dataset, thus making the graph fully connected.

4.4. TSP in the Real World [T2.4: 3 points]

In previous approaches, you have assumed that all graphs are fully connected. However, graphs that span in the real world often do not fulfil this requirement. As such, it is of the utmost importance to be able to provide a good approximation to the tsp problem in feasible time in these types of graphs.

In this task, you should then develop an algorithm that **always** gives a solution to the tsp problem even in graphs that are not guaranteed to be fully connected. The origin of the tsp must be provided as input by the user, which means that your algorithm must work with arbitrary starting points. If a solution is unfeasible because there is no path that returns to the origin and visits all nodes, your algorithm should output a message indicating that no path exists. If, however, a path does exist, then you should strive for your algorithm to give a tour as close to the optimal one as possible in feasible time. Even though that is the objective you should aspire to, this is a very hard task and as such, you should not lose yourself to the sole optimisation of this particular problem - the most relevant point in this assignment is to understand the complexity of tsp in real-world graphs, to be able to understand whether or not a tsp solution is possible and, when it is, to provide a tour that is not the most expensive one available. Your critical thinking skills will shine the most here and so you should discuss in your presentation the advantages and disadvantages of your proposed solution and explain your reasoning when you conceptualised your algorithm.

To do this, you should consider the given graphs solely as described in the dataset - this means that if an edge is not reported in the provided files, then you should assume it does not exist. The real-world graphs, in particular, are not fully connected originally and are sourced from real routes of ocean shipping and urban deliveries.

5. Demonstration & Presentation

Giving a presentation that is "short-and-to-the point" is increasingly important. As such, you are required to structure a short 15-minute presentation of your work, where you can highlight the most important aspects of your implementation.

[T4.1: 2.0 points] Prepare a presentation of your work with the main conclusions about your implementation of the TSP applied to the scenario of urban deliveries. Your presentation should focus on the explanation and analysis of the solutions you implemented in the aforementioned tasks. It is of the utmost importance for you to show that you thought critically about the algorithms that were implemented, namely where they showcase their advantages and situations in which they would be sub-optimal. In this discussion, you should also analyse the trade-offs between efficiency and optimality for all graphs provided in the datasets. The optimality of your heuristics can be analysed by comparing their results with the results produced by the backtracking algorithm. In addition to the heuristics, your presentation should also explain the implemented backtracking algorithm for the toy example graphs.

6. The Dataset

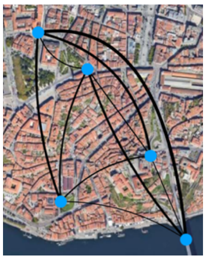
Unlike the first programming project, where you were given a dataset and your goal was to build a solution tailored to that dataset, in this second project the emphasis is on the algorithmic techniques and your interpretation of their outcomes. As such, you are given 18 graphs: 3 are small **toy examples**, 12 are medium-sized graphs for you to optimise your algorithms and the other three are scaled-down versions of real graphs generated from instances of real urban delivery datasets.

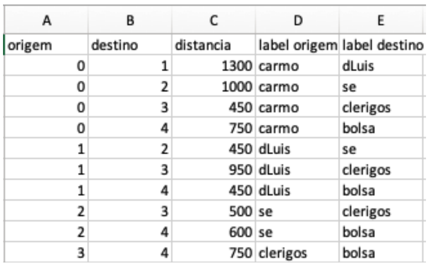
6.1. Toy Graph Examples

You are given three different small graphs, where backtracking strategies are feasible, allowing you to determine the ground truth for the solution and benchmark your heuristics. Note that the performance of your heuristics in the toy examples is not generalisable to the real-world datasets. Instead, these toy examples are given to support the development of your heuristics to have some degree of assurance that they are behaving as you expect them to.

The data is represented in different ways for the small and large graphs. The small graphs are described by a single file using a comma-separated values (csv) data format with nodes indicated by a numeric integer index (starting at 0) and explicitly listing all the edges as shown below. In this particular case, a small set of touristic interest points in the city of Porto are listed with an illustrative distance between them. An optional label column is also included which you can safely ignore.

As you can observe, in some cases the graphs are missing edges between nodes, i.e., they are not fully connected. This is perfectly normal as some routes or segments between nodes are to be discouraged and can therefore be assumed as infinite. You can use the graphs that are **fully connected** for your tests in **T2.2** and **T2.3** (stadiums.csv, tourism.csv) and the ones that are **not fully connected** (shipping.csv) for testing your solution to **T2.4** in comparison with the backtracking.



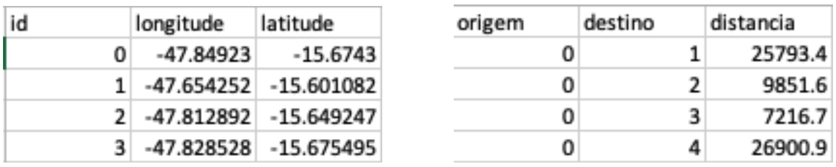


A	B	C	D	E
origem	destino	distancia	label origem	label destino
0	1	1300	carmo	dLuis
0	2	1000	carmo	se
0	3	450	carmo	clerigos
0	4	750	carmo	bolsa
1	2	450	dLuis	se
1	3	950	dLuis	clerigos
1	4	450	dLuis	bolsa
2	3	500	se	clerigos
2	4	600	se	bolsa
3	4	750	clerigos	bolsa

Fig. 1. Small graph example for points of interest in the city of Porto.

6.2. Real-world Graphs

For the larger 3 graphs, the representation includes a single file (named edges.csv) that lists the nodes along their geographic coordinates (in terms of longitude and latitude) and a second file (named edges.csv) that lists all the edges (assumed to be undirected) as a pair source-destination associated with the corresponding distance. The distance between nodes has been calculated using the Haversine distance, see Appendix A for a description. A sample of a tiny portion of the edges.csv file and nodes.csv file is shown below. You may use the geographic information to ensure these graphs are fully connected when requested (for **T2.1**, **T2.2** and **T2.3**). **Note** that your solution for **T2.4** must ignore the geographic information as the distances are already included for every feasible edge.



id	longitude	latitude
0	-47.84923	-15.6743
1	-47.654252	-15.601082
2	-47.812892	-15.649247
3	-47.828528	-15.675495

origem	destino	distancia
0	1	25793.4
0	2	9851.6
0	3	7216.7
0	4	26900.9

Fig. 2. Sample of large graph files (nodes.csv on the left and edges.csv on the right).

Regarding the 3 real-world graphs in this dataset we have the following edge and node statistics:

- Graph 1 has 1K nodes and ~500K edges
- Graph 2 has 5K nodes and ~4M edges
- Graph 3 has 10K nodes and ~10M edges

6.3. Extra Medium-Size Graphs

These 12 medium-sized and fully connected graphs have between 25 and 900 nodes so that you can experiment with your various algorithmic solutions. As with the real-world graphs, each of these graphs is represented exclusively by the edges, so as part of your pre-processing, you may need to determine the number of nodes by scanning all the edges first. The real number of nodes is included in the file name (edges_25.csv means the graph has 25 nodes).

7. Turn-In Instructions & Deadline

Submit a zip file named DA2023_PRJ2_G<GN>.zip on Moodle, where GN stands for your group number (e.g., 01_2), with the following content:

- Code folder, containing the program source code, but without compilation artefacts.
- Documentation folder, containing HTML documentation, generated using Doxygen.
- Presentation file (PDF format) that will serve as a basis for the demonstration.

Late submissions, up to 24 hours and 48 hours, will incur a penalty of 10% and 25% of the grade, respectively. No submissions will be accepted 48 hours after the deadline. Exceptions apply for justified and documented technical submissions issues. **Recall, the deadline is May 18, 2024 at midnight.**

Appendix A – Haversine Distance

General description in: https://en.wikipedia.org/wiki/Haversine_formula

C++ implementation: https://github.com/AhiyaHiya/haversine
Python: https://github.com/scikit-mobility/scikit-mobility/blob/master/skmob/utils/gislib.py
Pseudo-code using the atan function: (atan2 https://cplusplus.com/reference/cmath/atan2/)

```
convert_to_radians(coord):
   return coord*pi/180

Haversine([lat1, lon1], [lat2, lon2]):

   rad_lat1, rad_lon1, rad_lat2, rad_lon2 = convert_to_radians(lat1), convert_to_radians(lon1), convert_to_radians(lat2), convert_to_radians(lon2)

   delta_lat = rad_lat2 - rad_lat1
   delta_lon = rad_lon2 - rad_lon1

   aux = sin^2(delta_lat/2)  +  cos(rad_lat1) * cos(rad_lat2)  *  sin^2(delta_lon/2)
   c = 2.0 * atan2 ( sqrt(a),  sqrt(1.0-a))
   earthradius = 6371000 (in meters)

   return earthradius * c
```