Final project - Travelling Salesman Problem

Fanoarii Boyer - Jérémy Beaugeard - Arthur Guerineau - Adrien Le Saux

CIR3 - Graph Theory - Leandro Montero

30 April 2020

Contents

Chapter 1

Introduction

In this final graph theory project, we try to solve the Travelling Salesman Problem (TSP) using different algorithms and heuristics. Before we get to the code, it is important to consider how we are going to model the graphs.

In the TSP, we work with undirected complete graphs, which means they are very dense. Adjacency lists are useful for sparse graphs, but in our case it will be more appropriate, in terms of complexity, to use adjacency matrix[1].

To implement the adjacency matrix[2] we will use the C++ Boost[3] library. In this project we seek to compare the performance between different solutions to the same problem. As C++ is a low-level programming language it is very fast and will be suitable for our use. Moreover, the boost library is very well known and widely documented.

Chapter 2

Real-life situations

Chapter 3

Exact algorithm

3.1 Pseudo-code

3.2 Time complexity

3.3 Optimal Solution

3.4 Execution time and performance

Chapter 4

Constructive heuristic

4.1 Pseudo-code

Input: G is an undirected complete weighted graph with n vertices
Output: The optimal path found and the corresponding distance

```
finalPath ← empty
finalDistance ← ∞
foreach vertex V of G do
    path ← empty
    distance ← 0
    currentVertex ← V
    while there are still undiscovered vertices do
        minimumWeigth ← ∞
        foreach adjacent vertex V′ not discovered do
            w ← weight between V and V′
            if w < minimumWeight then
                minimumWeight ← w
                currentVertex ← V′
            end
        end
        path ← path + currentVertex
        Mark currentVertex as discovered
        distance ← distance + minimumWeight
    end
    returnToStart ← weight from nextVertex to V
    distance ← distance + returnToStart
    if distance < finalDistance then
        finalDistance ← distance
        finalPath ← path
    end
end
return finalPath and finalDistance
```

4.2 Time complexity

4.3 Optimal Solution

4.4 Execution time and performance

Chapter 5

Local search heuristic

5.1 Pseudo-code

5.2 Time complexity

5.3 Optimal Solution

5.4 Execution time and performance

Chapter 6

GRASP meta-heuristic

6.1 Pseudo-code

6.2 Time complexity

6.3 Optimal Solution

6.4 Execution time and performance

Chapter 7

Conclusion

Bibliography

[1] graphs - when are adjacency lists or matrices the better choice? Library Catalog: cs.stackexchange.com.

[2] The boost graph library - 1.72.0.

[3] Boost graph library: Adjacency matrix - 1.72.0.