Algorithm Conception and Analysis (CAL)
Master's in Informatics and Computation

Smart Cities Garbage Distribution: Graph approach on route planing and optimization

Final Report

Students:

Luís Cruz - UP201303248@fe.up.pt

Manuel Curral - UP201403027@fe.up.pt

Miguel Ramalho - UP201403027@fe.up.pt

April 4, 2019

Contents

1 Introduction

This report seeks to address the problem of optimizing a city's collection of trash through the monitoring of garbage containers. This is made possible by equipping said containers with volume sensors, therefore creating a way of knowing when a given container's content needs collecting.
A city's garbage containers layout consists of a set of points, or nodes, and a set of roads between each of them, also called edges. It can therefore be abstracted through the mathematical concept of a graph, namely a . Each node in the graph represents a container, a garbage truck's garage or a waste treatment plant. Each edge represents a route between two nodes.
This work contributes with a practical way of collecting trash using the most efficient route, going through as many full containers as it's volume permits, between it's garage and the waste plant.

2 Problem Formalisation

Previous to the formalisation of the problem at hand, there is a need to establish a formal notation to use throughout this paper, as follows:

- G represents the graph which abstracts an area's roads layout;
- G_p represents the result of preprocessing G;
- V represents the set of vertexes in graph G;
- V represents the set of vertexes in graph G_p;
- E represents the set of edges in the graph;
- V_i represents a specific node in the graph, referencing a physical location in a map;
- $P_{i,j}$ represents the path, a subset of V, connecting V_i and V_j.
- $In(V_i)$ represents the set of vertexes that go into V_i
- $Out(V_i)$ represents the set of vertexes that exit V_i

2.1 Input

The input to this model shall be a graph representation of the network which is to be analysed, as described before. Our data comes from the OpenStreetMap Project OSM. From this source, we extract an area's layout by using a tool, OSM2TXT Parser, that generates 3 files from a `.osm` file.
The first file contains information about the nodes: , latitude, longitude in degrees and radians. The second file contains information about the roads, ids, names and if it is two way road. Finally, the third text file contains information about the connection between two nodes, in other words, the information about the set of edges that defines a roads geometry.

2.1.1 Input Variables

When attempting to solve a sub-problem of the aforementioned garbage collection problem, there are a few specific variables to take into account, namely:

- Number of trucks;
- Number of containers;

2.2 Output

The output of the model is a sequence of nodes which corresponds to a path, optimal only in some cases, between the truck's garage and the waste treatment plant. There might be several criteria for choosing an optimal path, such as the number of trucks, number of containers full, the distances between each other and the containers' or trucks' volume. Thus, there are three possible definitions for the objective function.

2.3 Objective Function

2.3.1 One Truck and one Container

The first definition of an objective function, is finding a satisfactory route for a scenario of one truck and one container. With this criteria it is only needed to search for the shortest path from the garage to the container and the container to the treatment plant. See section 4.

2.3.2 One Truck and N Containers

A second definition of an objective function is finding a satisfactory route for a scenario where there is one truck and multiple containers. This scenario involves the search for a path that goes through all the containers, therefore disregarding the volume variable of the initial problem, and finding a good route for this subproblem. See section section 5.

2.3.3 N Trucks and N Containers

Finally, the third definition of an objective function is finding a satisfactory route for a scenario of multiple trucks and multiple containers. This function is the most complex one, since it involves choosing multiple paths in which different trucks can visit containers that were already visited by other trucks, but visiting those trucks decreases the travel distance. See section 6.

2.4 Graph Preprocessing

Given that many of the algorithms used to achieve a good truck route in this problem have a large time complexity, there was a need to alleviate the influence of the size of the graph, G, in its number of vertexes, as such an algorithm was implemented to reduce the effective number of vertexes and edges to be fed into said algorithms.
This algorithm iterates through each V_i in V from G and checks whether $|In(V_i)| = 1$ and if $|Out(V_i)| = 1$, and hiding this vertexes inside $E_{a,b}$ assuming $In(V_i) = V_a$ and $Out(V_i) = V_b$.

2.4.1 Pseudocode

Algorithm 1 Graph Preprocessing Algorithm

```
1:  procedure PREPROCESSGRAPH(Graph G)
2:      for i = 0 to |G → V| do                    ▷ Reset the hidden property
3:          V_i.hidden = false
4:      for i = 0 to |G → V| do                    ▷ Identify the redundant Vertexes
5:          if In(V_i) = 1 & & Out(V_i) = 1 then
6:              e ← newEdgeFromTo(In(V_i), Out(V_i))
7:              e.addHidden(EdgeFromTo(In(V_i), V_i))
8:              e.addHiddenVertex(V_i)
9:              e.addHidden(EdgeFromTo(V_i, Out(V_i)))
10:             In(V_i).removeEdgeTo(V_i)
11:             Out(V_i).removeEdgeFrom(V_i)
12:             In(V_i).addEdgeTo(Out(V_i))
13:             Out(V_i).addEdgeFrom(V_i)
14:             V_i.hidden ← true
15:     G_p = EmptyGraph()
16:     for i = 0 to |G → V! do                    ▷ Construct the preprocessed graph
17:         if V_i.hidden = false then
18:             G_p.addVertex(V_i)
19:     Return G_p
```

2.4.2 Time Efficiency

Since all the operations of removing and inserting edges can be done in linear time, due to an incremented memory cost, the time complexity of this algorithm is determined by the three **for**s in pseudocode 1 since none of the is nested all the instructions are repeated $|V|$ times:

$$O(|V| + |V| + |V|) = O(|V|)$$

2.4.3 Space Efficiency

Since there is an extra need, from the original graph implementation, to save the edges that go into a vertex and the ones that exit it the space complexity will be:

$$O(|V| + |V| + |V|) = O(|V|)$$

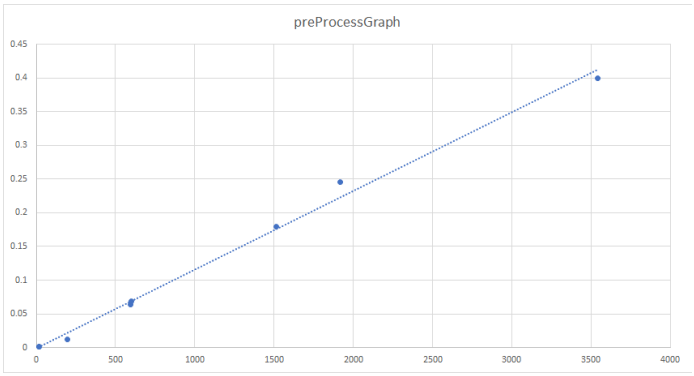

Figure 1: Empiric time efficiency for the preprocessubg algorithm

2.5 Floyd-Warshall (with path reconstruction)

This algorithm is a dynamic programming algorithm used to calculate the shortest path between every pair of vertexes on a graph, this version of the algorithm (path reconstruction) allows us to reconstruct the shortest path, $P_{i,j}$ between two vertexes, V_i and V_j [1].

2.5.1 Pseudocode

Algorithm 2 Floyd-Warshall

```
1: let dist be a |V| × |V| array of minimum distances initialized to ∞
2: let next be a |V| × |V| array of vertex indices initialized to null
3: procedure FloydWarshallWithPathReconstruction
4:     for each edge(u, v) do
5:         dist[u][v] ← w(u, v)                ▷ the weight of the edge (u,v)
6:         next[u][v] ← v
7:     for k = 1 to |V| do                     ▷ Floyd-Warshall implementation
8:         for i = 1 to |V| do
9:             for j = 1 to |V| do
10:                if dist[i][j] > dist[i][k] + dist[k][j] then
11:                    dist[i][j] ← dist[i][k] + dist[k][j]
12:                    next[i][j] ← next[i][k]
13: procedure getPath(u, v)
14:     if next[u][v] = null then
15:         return[]
16:     path = [u]
17:     while u ≠ v do
18:         u ← next[u][v]
19:         path.apend(u)
        return path
```

2.5.2 Time Efficiency

As we can see in pseudocode 2 there are three nested loops which iterate $|V|$ times, since this is the highest impact operation it defines the time efficiency factor of the algorithm and so, as expected for the Floywd-Warshall algorithm, the time efficiency will be:

$$O(|V|^3)$$

2.5.3 Space Efficiency

To run this algorithm we need to create to new containers that amount to a size of $|V| \times |V|$ each, in this way the space efficiency of this version of the algorithm is:

$$O(|V| \times |V| + |V| \times |V|) = O(|V|^2 \times |V|^2) = O(|V|^2)$$

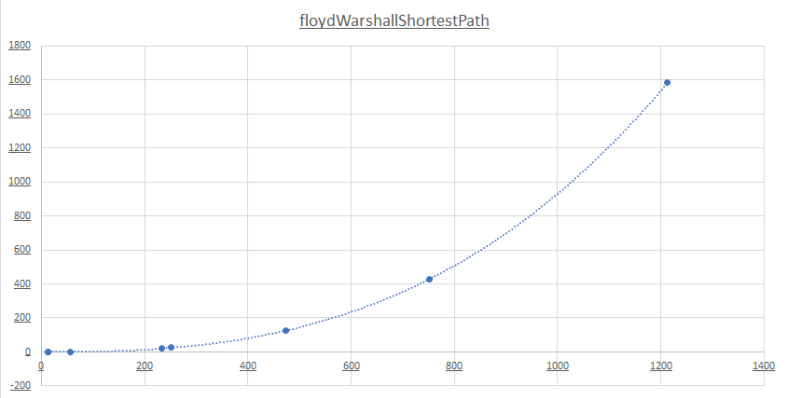

Figure 2: Empiric time efficiency for the Floyd-Warshall algorithm

2.6 Nearest Neighbour

This algorithm is a constructive heuristic used to find a path through a set of nodes using a simple behavior: In each node we calculate the next one by finding the closest one from the set of unvisited nodes. This is used for its efficiency in finding a plausible path that is, in average, between the shortest route and a length of 25% of its size [2].

2.6.1 Pseudocode

Algorithm 3 Nearest Neighbour

```
1: procedure NearestNeighbour(Vertex P)
2:     queuenodesSorted        ▷ The remaining nodes sorted by distance
3:     result = []
4:     V_t ← V                 ▷ Copy the list of vertexes into a temp one
5:     while |V_t| > 0 do
6:         nodesSorted.resortRelativeTo(P)
7:         result.append(P)
8:         P ← result[0]
9:         nodesSorted.removeTop()
       return result
```

2.6.2 Time Efficiency

Given that our implementation of the *resortRelativeTo* has a time efficiency of $V \times \log_2 |V|$ [3](stl function), and this instruction is the only one without linear efficiency inside the while loop in Algorithm 3 the resulting time efficiency will be

$$O(V \times (V \times \log_2 |V|)) = O(V^2 \times \log_2 |V|)$$

2.6.3 Space Efficiency

To run this algorithm we need to create a single container that amounts to a size of $|V|$, in this way the space efficiency of this version of the algorithm is:

$$O(|V|)$$

3 Class Diagram

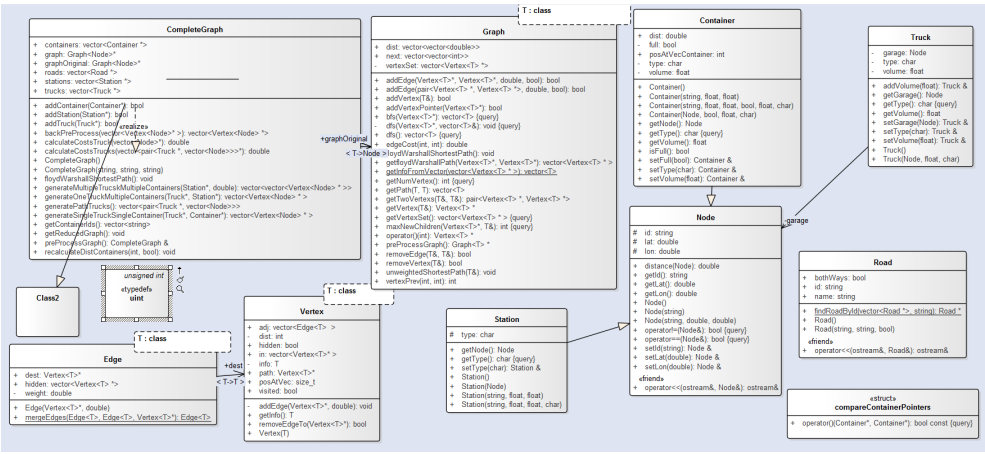

Figure 3: Example output from the created script using Google Maps API

4 Approach 01 - One Truck and one Container

This problem formalization is the simplest one. Essentially, the problem is falls back to considering a single truck, that starts in a given garage, a single container and a single delivery station.
Since this approach can be tested after the graph has been preprocessed and gone through the Floyd-Warshall algorithm, the optimal path for this scenario is to get the shortest path from the garage to the container (using the getPath function from Floyd-Warshall) and repeating the same process from that container to the final station.

Since there is only one possible order for the nodes to be iterated and the shortest path is used in every transaction this path is the optimal solution for the problem.

5 Approach 02 - One Truck and N Containers

On a second approach, we decided to consider the existence of a single truck with an hypothetical infinite volume and then we considered the existence of N containers that need to be collected by this truck. To solve this problem we used a constructive heuristic, Nearest Neighbour algorithm 2.6, so that given an initial garage for the truck, a list of Vertexes to visit, the containers, and a final Station we can find a deterministic path by always navigating to the closest container (until all of them are visited) and then by doing the same for the station.
Since this algorithm is executed after the Floyd-Warshall we can guarantee that the path used to go from one vertex to another is always the shortest path.

6 Approach 03 - N Trucks and N Containers

Finally, we added a few more restrictions to the previous formalization. Namely we set that each truck has a maximum volume and will, at times, be insufficient to collect every container. So, this third approach will handle multiple containers and multiple trucks, creating a path for each of them. To distribute the containers accordingly we execute a process similar to the one presented in section 5, that is we use the nearest neighbour heuristic for a truck until it is full and then we dispatch it to the final station and, if there are still containers left to collect, we repeat the process for a new truck to guarantee that every container is picked up.

6.1 Algorithm

6.1.1 Pseudocode

Algorithm 4 Nearest Neighbour for multiple trucks

```
1: procedure NearestNeighbourTrucks(Containers c)
2:     result = []
3:     V_t ← V                          ▷ Copy the list of vertexes into a temp one
4:     while |c| > 0 do
5:         Truckt                                              ▷ A truck instance
6:         t.path = NearestNeighbour(t.garage)   ▷ with stop condition for volume
7:         result.append(t)
       return result
```

6.1.2 Time Efficiency

Considering V_{cont} to be the set of Vertexes that represent containers, this algorithm will have a time efficiency of:

$$O(|V_{cont}| \times O(NearestNeighbour)) = O(|V_{cont}| \times (V^2 \times \log_2 |V|))$$

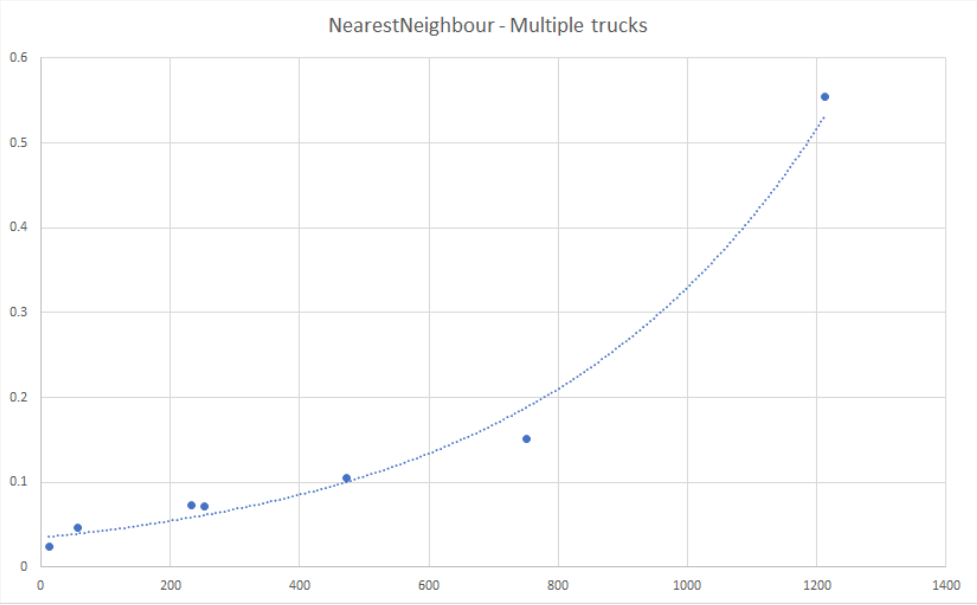

Figure 4: Empiric time efficiency for the Nearest Neighbour for multiple trucks

7 Results

Depending on the type of objective function defined by the user, the output should be a path in which the resultant the truck or trucks starts from the garage and goes through each container ending the trip at the assigned station. In the example above shows a case where there is a truck for multiple containers.

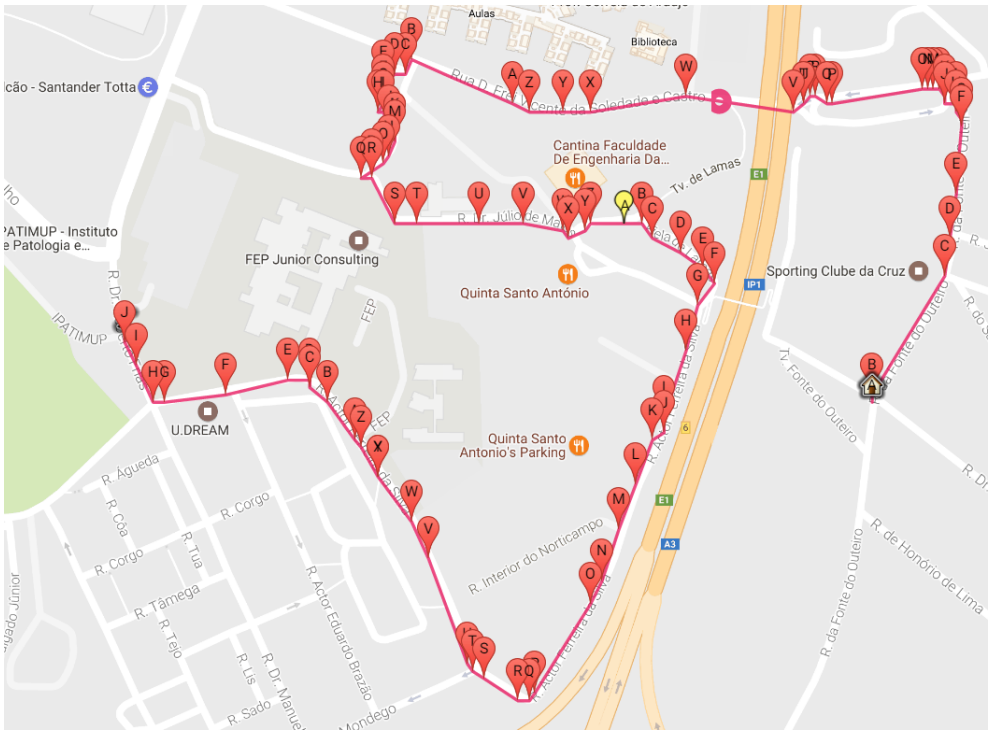

Figure 5: Example output from the created script using Google Maps API

8 Conclusion

Concluding, the aim of this paper was to discuss a possible approach to the problem of planning the route of a network of trucks connecting two points, the truck's garage and the waste treatment plant. A path is considered to be sufficient if it meets one of the three objective functions. This possible instances of the objective functions were discussed in previous sections. This problem was reduced to a shortest path algorithms, and a unique type of trash.

9 References

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990). Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill. ISBN 0-262-03141-8. See in particular Section 26.2, "The Floyd–Warshall algorithm", pp. 558–565 and Section 26.4, "A general framework for solving path problems in directed graphs", pp. 570–576.

[2] Johnson, D. S.; McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF). In Aarts, E. H. L.; Lenstra, J. K. *Local Search in Combinatorial Optimisation*. London: John Wiley and Sons Ltd. pp. 215–310.

[3] `http://www.cplusplus.com/reference/algorithm/sort/`. Accessed on April, 2017.