OPTIMIZATION
ASTAR ASSIGNMENT – A ROUTING PROBLEM

Authors: Lawrence Adu-Gyamfi & Alexis Oger

Date: 24/12/2018

Table of Contents

1. INTRODUCTION

The objective of this report is to present in details our implementation of the Astar algorithm to the routing problem of finding the the optimum path between two points.- Bas´ılica de Santa Maria del Mar (Pla¸ca de Santa Maria) in Barcelona to the Giralda (Calle Mateos Gago) in Sevilla-in this case.

1.1 Problem Statement

The assignment consists in computing an optimal path (according to distance) from Bas´ılica de Santa Maria del Mar (Pla¸ca de Santa Maria) in Barcelona to the Giralda (Calle Mateos Gago) in Sevilla by using an AStar algorithm.

As the reference starting node for Bas´ılica de Santa Maria del Mar (Pla¸ca de Santa Maria) in Barcelona we will take the node with key (@id): 240949599 while the goal node close to Giralda (Calle Mateos Gago) in Sevilla will be the node with key (@id): 195977239.

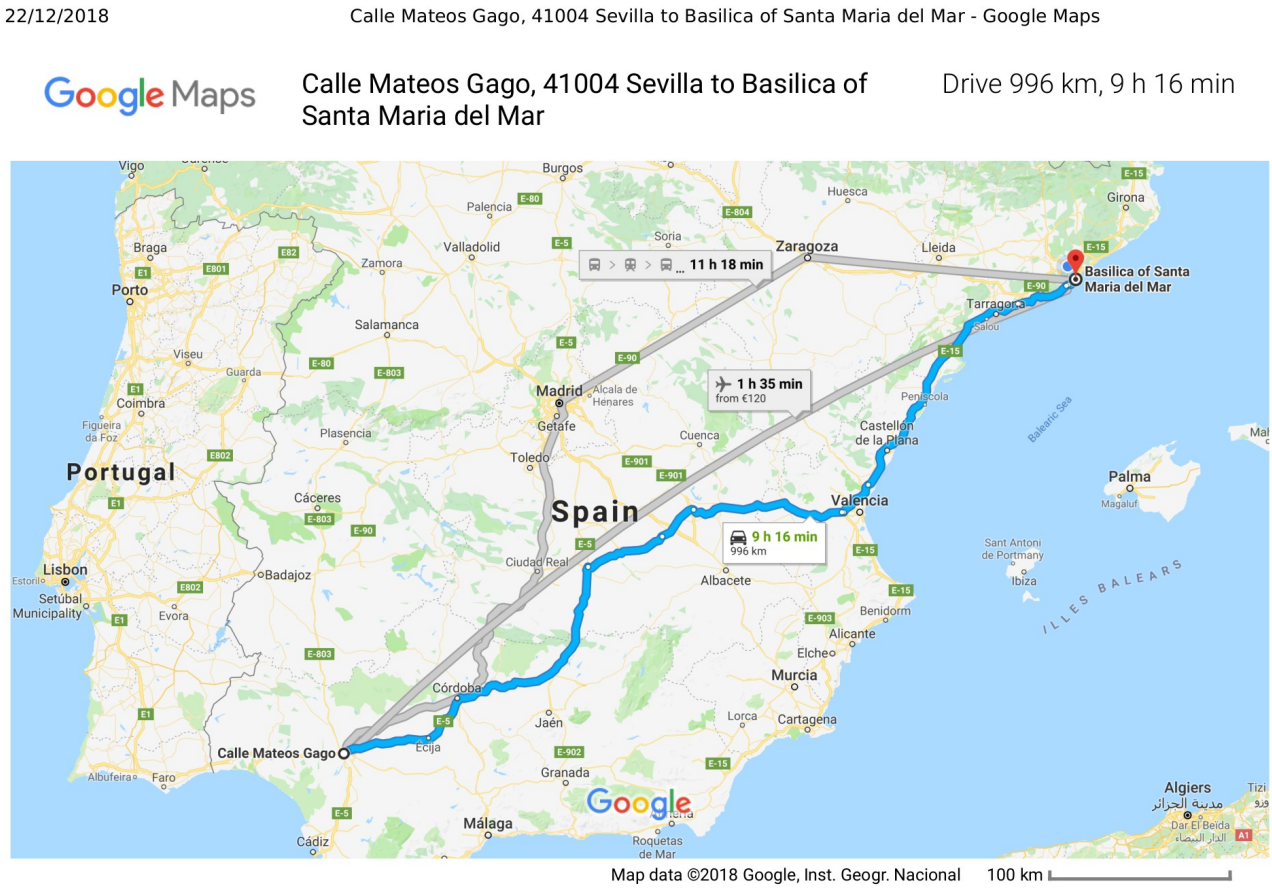

Fig 1.1 :Route Bas´ılica de Santa Maria del Mar and Giralda as per google maps.

1.2 Description of Algorithm

As noted in the problem statement, the solution has been implemented using the Astar algorithm. The alrgorithm is described below.

The Astar algorithm is part of a family of algorithms called **best-first search** algorithms which are are typically implemented for path finding problems in combinatorial algorithms. These types of algorithms explore a search space (or graph to be specific) by expanding on the most promising node which is decided on based on a specified rule known as the heuristic evaluation function.ref[2] Our version of the heuristic function for this problem is explained in section 2.

The general structure of a typical Best-First Search Algorithm is shown below ref[2]

1. Put the start node (S) on a list called OPEN of unexpanded nodes.
2. If OPEN is empty, exit with failure; no solution exists.
3. Remove from OPEN a node *n* at which *f (the result of the heuristic evaluation and the distance from the source node)* is minimum, and place it on a list called CLOSED to be used for expanded nodes.
4. Expand node *n*, generating all its successors with pointers back to *n.*
5. If any of *n*'s successors is a goal node, exit successfully with the solution obtained by tracing the path along the pointers from the goal back to S.
6. For every successor n^i of *n:*
   a) calculate f(n^i).
   b) If n^i was neither on OPEN nor on CLOSED, add it to OPEN. Attach a pointer from n^i back to *n.* Assign the newly computed $f(n^i)$ to node n^i .
   c) If n^i already resided on OPEN or CLOSED, compare the newly computed $f(n^i)$ with the value previously assigned to n^i . If the old value is lower, discard the newly generated node. If the new value is lower, substiture it for the old (n^i now points back to n instead of to its previous predecessor). If the matching node n^i *resided on CLOSED, move it back to OPEN.*
7. Go to step 2.

A-star algorithm incorporates the distance from the start node in addition to the estimated distances to the goal, and consequently cannot be considered as a greedy algorithm.
In each evaluation, the Astar algorithm chooses the path that minimizes the the following:

$$f(n) = g(n) + h(n)$$

n – *the current node*
f(n) - *testimated distance from source node to goal node going through the node n*
g(n) – *the cost or distance from the source node to the node n.*
h(n) - *the cost evaluated by the heuristic function estimating the cost from n to the goal node.*

The choice of the heuristic funciton is determined based on the specific problem. The heuristc function used in our implementation is explained further in the section 2.
The pseudocode of the A*star algorithm being a specialised form of the best-first algorithm is shown below:.

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17    }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

2. METHODOLOGY

This section explains our code as used in our implementation of the program. It also explains the heurestic evaluation function that has been considered in the implementation.

2.1 The Code of the Program

Our implementation has been done using the C language. The executable of createbin.c requires two arguments; the name of the csv file to read and the name of the binary file to be created.
Firstly it opens the csv file and count the number of lines starting with the letter «n». This number is equal to the number of nodes of the graph. Then it allocates the required memory and initialize the list of nodes.
Then we go back to the start of the file. For each line starting by «node» we store the id,latitude,longitude in the next free node of the list.
Afterwards, for each line starting by «way», for each couple of nodes linked:

- we use the binary search to find their index(node1 and node2)
- we call the function addway(node1, node2)
- if the way is dual, we call addway(node2, node1)

The function addway adds the index of the successor node2 in the list of successors of node1. It performs a realloc for the memory if it is necessary.
Finally the graph had been created and we can perform astar. But in order to use this graph many times without reading the csv file each time, we will store the graph in a binary file.

The code of the A-star algorithm together with its helper functions for dealing with the open list is written separately in astar.c. The heuristic distance evaluation is also specified in a separate function in a separate file a_star_distance..
The code in *"readbin .c"* implements all the program modules and interfaces. It starts by reading the binary file with all the information as stored previously. The required memory is allocated for the nodes and the successors. The nodes are read and the successors are pointed to each node.
The A-star algorithm is then run and the output of nodes and their distances are saved to an output file.

2.2 Memory Models

2.2.1. Node

Below is the structure that has been used to store the details of each node:

```
typedef struct node{
        unsigned long id;
        //char name[184];
        double lat, lon; // latitude and longiture of node
        unsigned short nsucc, successors_size; //list of successors and memory allocated to list
        int* successors; // number of successors
        double g, h; // g= actual distance from source node to node, h= heuristic distance to goal node
        char status;
        struct node* previous, next, parent;
}node;
```

2.2.2. Open List

The open list has been implemented using linked list data structures as proposed. This was very useful in easily accessing the node with the smallest distance (f), as this was always the first one in the list.
A character variable named “status” has been used in the node structure to track whether the node has not been visited, is in the open list or in the closed list.
Below is the structure declaration for the open list:

```
// Structure datatype for list
typedef struct {
   node *top;
   node *bottom;
   unsigned long size; // number of nodes in list
}list;
```

Separate helper functions have been developed to insert a node in the list based on the (f) and to remove a node from the list.

2.2.3. Heuristic Distance

The heuristic distance from each node to the goal node is extimated using the harvesine formula to calculate the great-circle distance between the two nodes. This estimation is done using the corresoponding longitudes and latitudes of the two points.
The version of the harvesine formula implemented is shown below.

$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

D is the distance between the two points (along a great circle of the sphere
r is the radius of the sphere. Which is the eart’s radius in this case (mean radius = 6,371km);

λ1, λ2: longitude of point 1 and longitude of point 2.

φ1,φ2: latitude of point 1 and latitude of point 2

2.3 How to Execute the Code

The instructions for executing the code are as follows:

1. Create all the executables by issuing the make command

   $ make

2. Create the binary file with "create", "spain.csv" and "<output>.bin" file as output

   $ ./create spain.csv <output>.bin

3. Find the path with "solve" and the binary output file

   $ ./solve <output>.bin

3. RESULTS

This section presents a summary of the results of running the entire module of the program including:

- Reading of the map data from the CSV file
- Creation of the graph and storage into a binary file.
- Reading of the graph from the binary file
- Finding the optimum route with the A- star function

The program has been run on a computer with the following charateristics:

- Computer Brand: Apple MacbookPro 11,1
- CPU: Intel(R) Core(TM) i5-4308U CPU @ 2.80GHz
- Memory – 8GB
- Operating sytem: Ubuntu 18.04.1 LTS
- Compiler: GCC 7.3.0

3.1 Results of Running the Program

3.1.1.Metrics of the Program

Time spent to read CSV: **2.124868 s**
Time to create graph: **46.798506 s**
Time to create binary file: **36.972083 s**
Time spent to read binary fileL: **2.107804 s**
Time spent to find optimum distance with A star: **9.438836 s**
Optimum distance = **958815.012135 m**

3.1.2. Final Route

For the sake of clarity and due to the length of the final route generated by the program, only a section of the results is shown below (highlighting the source node and the goal node, together with the total distance obtained)
The full list of the path from the source node to the goal node has been attached to this report separately.(results_final.txt)

Node id:240949599 | Distance:0.000000
Node id:240944785 | Distance:26.926933
Node id:240936347 | Distance:40.716571
Node id:240936348 | Distance:82.863623
Node id:30647274 | Distance:101.262493
Node id:240939090 | Distance:106.755711
…
Node id:195977233 | Distance:958736.719304
Node id:195977234 | Distance:958743.868538
Node id:195977236 | Distance:958785.826246
Node id:1344516329 | Distance:958802.893998
Node id:710632992 | Distance:958805.556372
Node id:195977239 | Distance:958815.012135

4. CONCLUSIONS

As part of this assignment and as presented in this report, we have implemented the A-star algorithm to solve a typical routing problem of finding the the optimum path between Bas´ılica de Santa Maria del Mar (Pla¸ca de Santa Maria) in Barcelona to the Giralda (Calle Mateos Gago) in Sevilla-in this case.
Comparing our optimum distance to that provided by google maps, for instance(shown below), we realise that our implementation yields a lower distance **(959 km).**

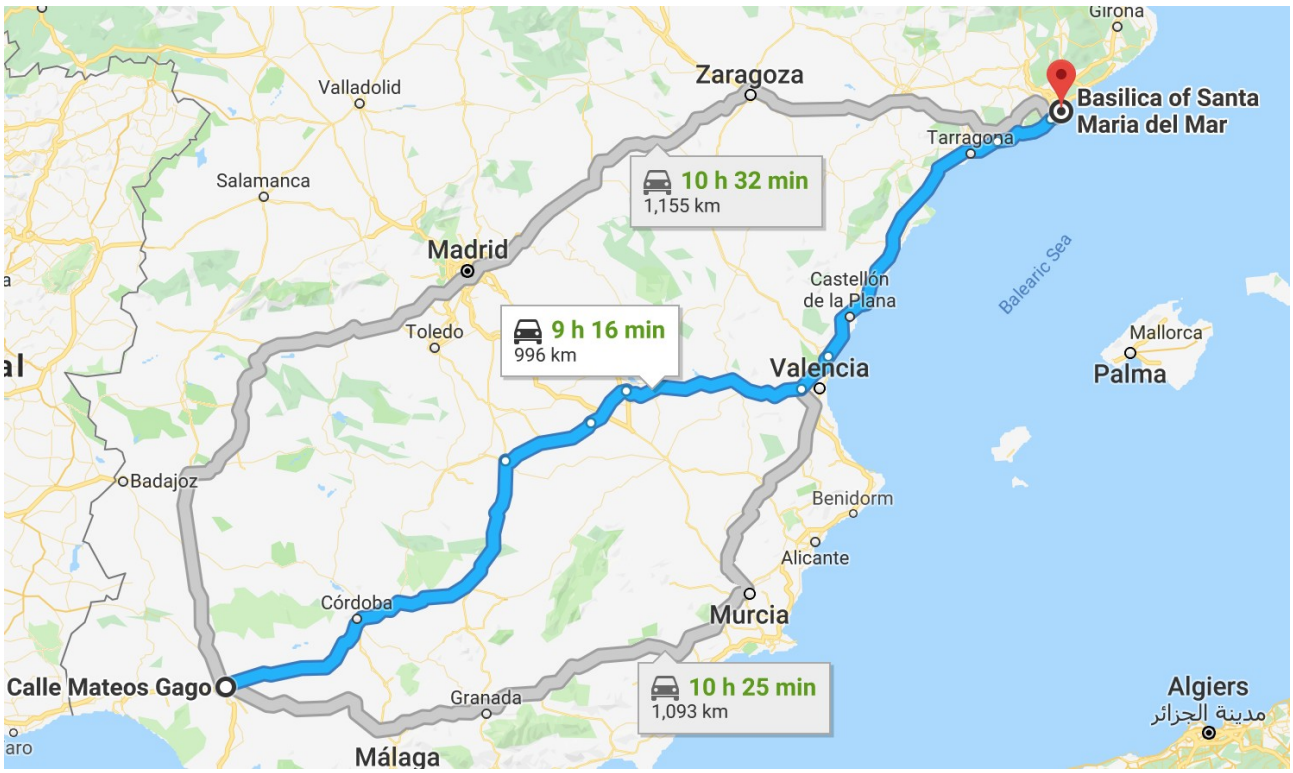

Fig 4.1: Distance approximation from google maps

And this is as expected as google maps optimizes the route based on the minimum time while our version was more focused on the route with the optimum distance.
For the purpose of this assignment and implementation only two variations of distance measurement have been considered (the harvesine distance and the equirectangular approximation). The harvesine distance approximation has been used finally in our implementation and this does yield good results.

5. REFERENCES

The following references have been used in one way or the other in the preparation of the solution outlined in this report. The have been referenced as and when applicable.

1. https://en.wikipedia.org/wiki/A*_search_algorithm
2. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Judea Pearl
3. https://www.movable-type.co.uk/scripts/latlong.html
4. https://www.google.com/maps/dir/Calle+Mateos+Gago,+41004+Sevilla/Basilica+of+Santa+Maria+del+Mar,+Pla%C3%A7a+de+Santa+Maria,+1,+08003+Barcelona/@39.5144827,-2.8038059,7z/data=!4m13!4m12!1m5!1m1!1s0xd126c1bd01b2edf:0x2a28485183a313e5!2m2!1d-5.990843!2d37.3861077!1m5!1m1!1s0x12a4a2fe5e0da457:0xb4bc945c1fc0e160!2m2!1d2.1820711!2d41.3838871