Optimisation

Determination of the shortest path from Barcelona to Sevilla using A* Algortihm

DAVID ARÀNEGA SEBASTIÀ
MOHAMED TAYBI BOUTAHRI

Universitat Autònoma de Barcelona - 8th of January of 2020

1 Introduction

Maps enable commuter to organize information relating to the person 's current location and the person's desitnation. A commuter travelling from A to B in a particular geographical space is aware that network of cities are linked together by a communication channels (routes or roads) from his source to his destination. In the light of the above, there are several routes of varying magnitude from where he could choose from. Based on the above explanation, one can conceptualize map of a given geographical location as a graph representing a group of nodes called vertices and links or routes between them, technically known as edges.

Given a network of cities and distances between them, a commuter travelling from one city to another needs to take cognizance of the distance taken and cost incurred from his source to his destination. This invariably, results into a problem of determination of the shortest possible path so as to optimize the available resources described in terms of distance and cost. One practical way of handling this kind of problem is the understanding and application of graph theory algorithms which are capable of handling shortest path problems.

The aim of this project is to develop an implementation of the A* algorithm and use it to find the shortest path between the Basílica de Santa Maria del Mar (Plaça de Santa Maria) in Barcelona and Giralda (Calle Mateos Gago) in Sevilla.

We have divided the implementation in two different programs, one that reads the information from the database and generates a binary file with the graph information properly written and another one which uses the generated binary file and applies the A* algorithm.

The data used to generate the graph over the algorithm we will work with, has been obtained from openstreetmaps.com and a map containing the data for Spain has been used.

2 Treatment of the data

The data comes in written as a .csv file with ”—” as a delimiter for the different values. The first three rows of the file are useless and must be manually removed in order for our code to work properly. The file has three types of fields: node, way and relation. The structure of each field is:

```
node|@id|@name|@place|@highway|@route|@ref|@oneway|@maxspeed|node_lat|node_lon
way|@id|@name|@place|@highway|@route|@ref|@oneway|@maxspeed|membernode|membernode|membernode|...
relation|@id|@name|@place|@highway|@route|@ref|@oneway|@maxspeed|rel_type|type;@id;@role|...
```

Storage of the data

The data for the binary file will be stored as an array of structures with the following format:

```
typedef struct{
unsigned long id; //Number that identifies each node
char *nombre; //Name of the node.
double longitud, latitud; //Latitude and longitude of the node.
unsigned short n_sucesores; //# of successors of the node.
unsigned long *sucesores; //Pointer to a vector with id’s.
short estado; //Will indicate the QUEUE in which
//the node is for the A* algorithm.
}nodo;
```

Reading of the data

The best way of reading the file is to read it line by line as a string with the function getline and then split the line at the places with a ’—’ with a “tokenizer” such as *strsep*. In this framework the fields are strings. The numeric ones must be effectively converted to numbers with the standard functions strtoul and atof.
´

In order to properly get all the information from the database the script takes the following steps.

1. First loop to obtain the number of nodes in the file.
2. Allocates the memory for the array of structures.
3. Second loop to get the id, *nombre, longitud and latitud for each node.
4. Loop that counts the number of successors each node has.
5. Allocates the memory for the vector of successors of each node.
6. Final loop that fills the vectors of successors we have just generated.

With the data properly allocated only remains to generate the binary file and follow up with the A* algorithm.

3 Implementation of the algorithm

For the A* implementation, on top of the nodo structures previously mentioned, the following data structures have been:

```
typedef struct{
nodo *padre, *nodo_i;
double coste, coste_h;
}estado_nodo;
```

This structure contains all the necessary information for a node that has been visited. It will be created and destroyed as the node enters or leaves the open and closed queues. The first two pointers of the structures are a pointer to the father node and the node itself respectively. The two doubles are the g(n) and f(n) functions.

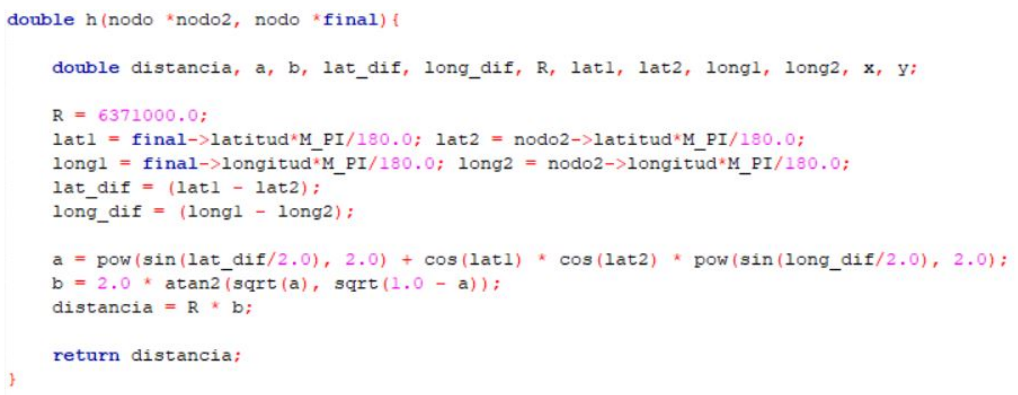
```
typedef struct{
estado_nodo estado;
struct listado *siguiente;
}estado_nodo;
```

This last structure is the one that defines the linked lists used in the script for the open and closed lists.

In order to work with the linked lists several functions have been defined such as to add, delete and extract the position of different elements.

The heuristic function has been defined as the straight distance between two points through the surface of the earth.

The heversine formula has been used in order to compute this value.



```
double h(nodo *nodo2, nodo *final){

    double distancia, a, b, lat_dif, long_dif, R, lat1, lat2, long1, long2, x, y;

    R = 6371000.0;
    lat1 = final->latitud*M_PI/180.0; lat2 = nodo2->latitud*M_PI/180.0;
    long1 = final->longitud*M_PI/180.0; long2 = nodo2->longitud*M_PI/180.0;
    lat_dif = (lat1 - lat2);
    long_dif = (long1 - long2);

    a = pow(sin(lat_dif/2.0), 2.0) + cos(lat1) * cos(lat2) * pow(sin(long_dif/2.0), 2.0);
    b = 2.0 * atan2(sqrt(a), sqrt(1.0 - a));
    distancia = R * b;

    return distancia;
}
```

4 Results

The best path we've managed to obtain has a length of 965.840 km, far away from the actual optimal one from the professor's solution.

This number comes as a result as using a weight of $k = 1.21$ to the heuristic function.

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

If no weight was used, the algorithm takes far too long for it to be practical. If a higher weight is used, the result gets worse.

k	Time(s)	Recorrido(km)
1.21	54.73	965.840
1.35	0.54	982.341
1.5	5.05	1096.941
1.6	0.26	1098.186
1.9	0.11	1070.268

An increase in time performance generally means a decrease in optimality.

We assume that a better performing heuristic could have been used on top of a better optimized code in order to obtain a better result for the algorithm.

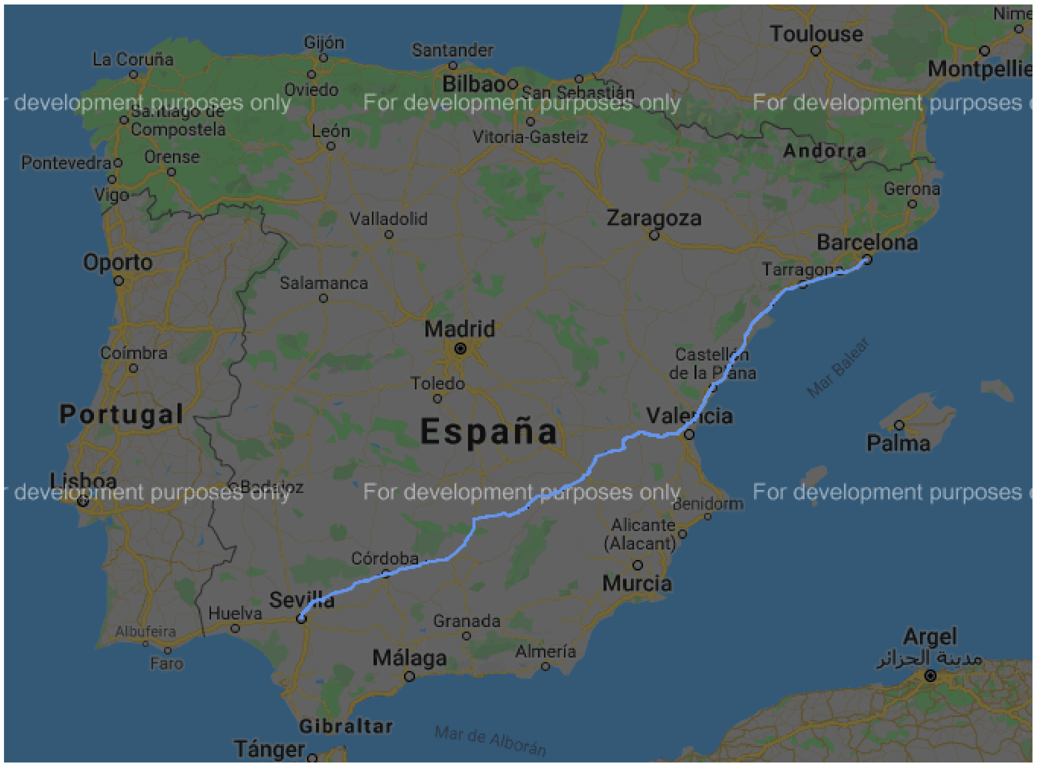


Figure 1: Graphic result of our optimal path

5 Conclusions

To find the optimal routing paths to a specific destination remains a practical use for commuters, especially when traveling a new geographical area for the first time. In this project, we have successfully developed and implemented an algorithm which finds the shortest path between Barcelona and Sevilla. The search for the shortest path was achieved by the Astar, simulated using C programming code. This project corroborate the application of the Astar algorithm to solve real-life problems.

A-star (A*) is a mighty algorithm in Artificial Intelligence with a wide range of usage. However, it is only as good as its heuristic function(which can be highly variable considering the nature of a problem).That's why A* is the most popular choice for pathfinding because it's reasonably flexible.