ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Artificial Intelligence
Project I:
Intelligent Taxi Service

January 8, 2019

Submitted By:
Beyer, Michael

Contents

1 Introduction

Goal for this project was implementing the core infrastructure for an intelligent taxi service similar to known apps like *Uber* or *Beat*. Based on the clients GPS-location the shortest path to the closest available taxi had to be found using the A*-Algorithm.

For more information and a better overview the source code and all generated path-kml files can be found enclosed to this report.

2 Implementation

2.1 Software structure

In figure 1 the structure of the programs is represented by a flow chart. First the relevant data from the client, taxis and map nodes csv-files are loaded into lists without any preprocessing. After building the map the A*-algorithm searches for paths to the given problem and, if successful, a KML-file with all found paths in the *Routes* subdirectory is created.

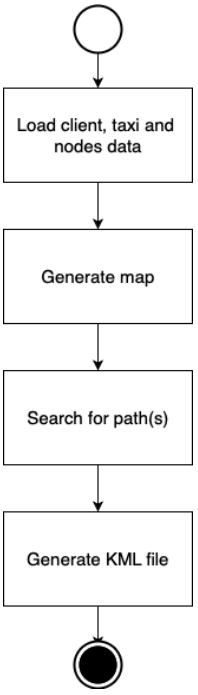

Figure 1: Software structure

The application can be started with the following command:

```
java -jar AI_TaxiService.jar client.csv taxi.csv nodes.csv outputFileName
```

2.2 Map creation

The map for the given coordinates is represented by an undirected graph. First all nodes are created and connected with each other according to their road id, afterwards crossings between roads are added using a hashmap to find nodes with the same coordinates.

The A* algorithm in this implementation compares the coordinates of its current node with the goal nodes and only if they match a valid path is found. Since taxis (goals) and client (start) positions don't necessarily match the coordinates of nodes in the map, a KD-search-tree is used to find the nearest neigbour(s) on the map for those nodes.

2.3 A* algorithm

Finding the shortest path(s) is done by a slightly modified A* algorithm. Each node the algorithm visits is ranked by computing the estimated cost from the node to a goal according to (1)

$$f(n) = g(n) + h(n) \tag{1}$$

where n is the current node, *g(n)* is the cost from the start node to the current node and *h(n)* is a heuristic function that estimates the cost to the goal. Using an admissible heuristic guarantees that the algorithm will return the shortest path possible.

The *AStar* class offers two different implementations for the distance estimation function: Euclidian distance or Manhattan distance. In this case the euclidian distance is used, since it will never overestimate the cost to the goal and thus is admissible.

Instead of returning the first found path like the "classical" A* algorithm would, this algorithm keeps searching for additional paths as long as the heuristic cost of the search frontier is lower or equal to the cost of the already found path(s).

3 Examples

3.1 Route for provided taxi and client locations

With the provided csv files the taxi number 100 is selected as the closest taxi. The generated path for this configuration can be seen in figures 2 and 3, where the client is colored orange and taxis are colored blue.

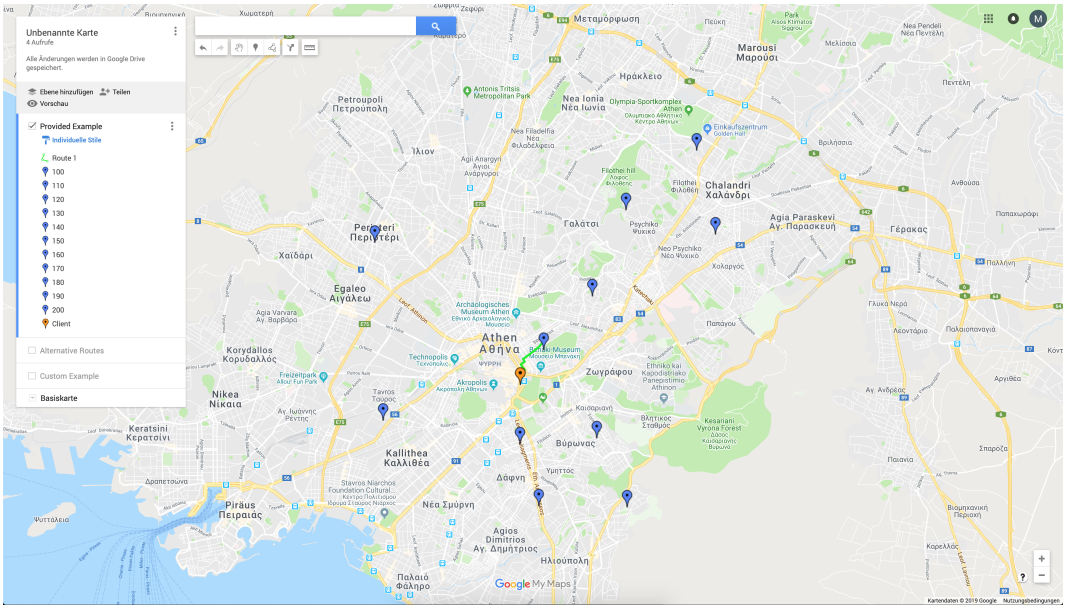

Figure 2: Route for the provided client and taxi locations

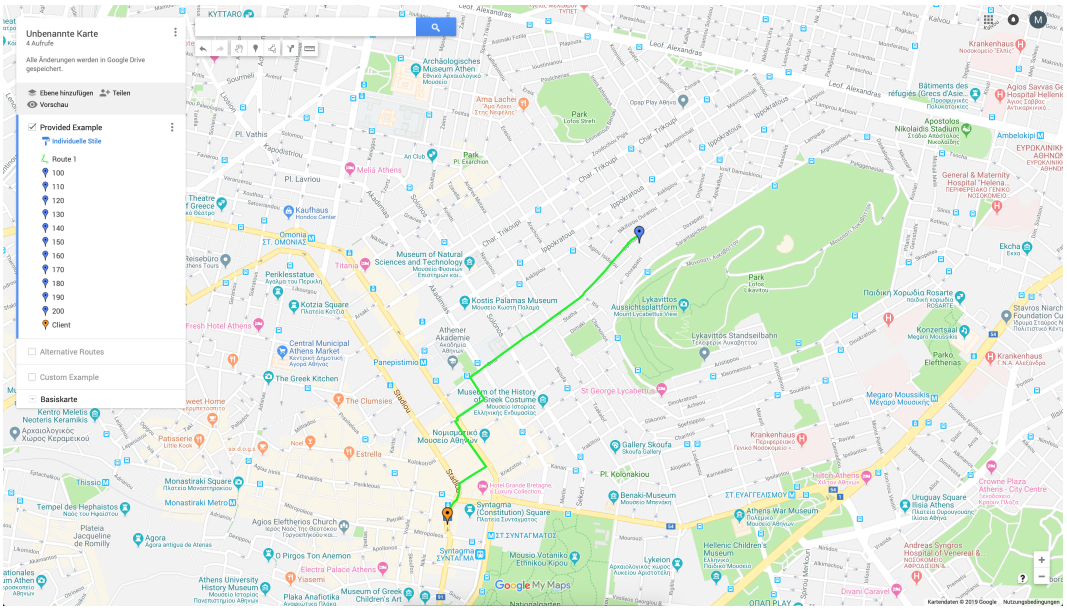

Figure 3: Close up of the route for the provided client and taxi locations

3.2 Route for custom taxi and client locations

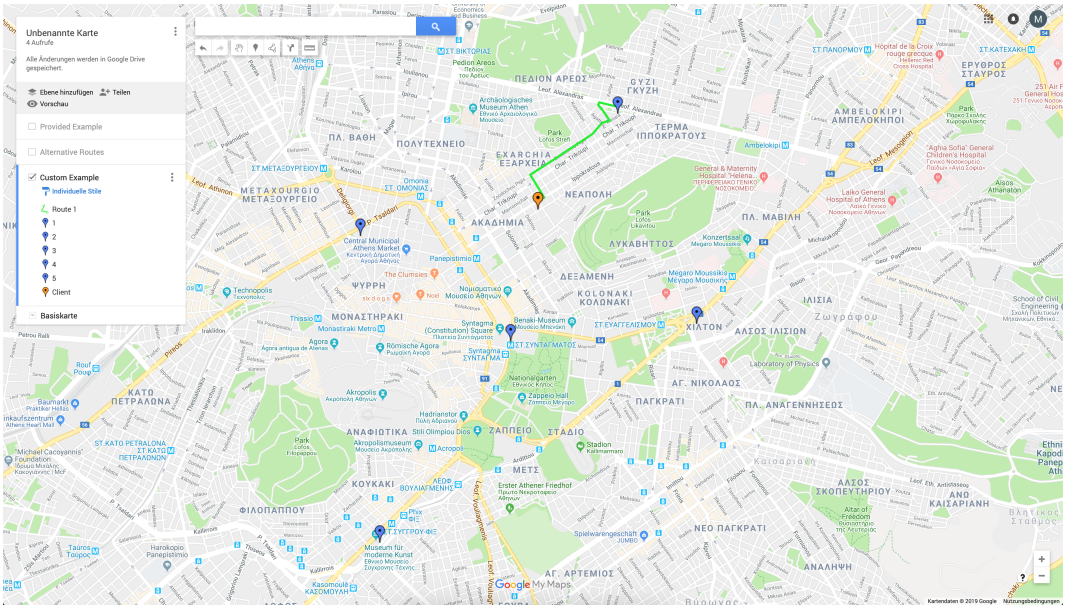

Figure 4: Route for custom client and taxi locations

3.3 Suggesting alternative routes

A simple map (with no real use case) was created to show the algorithms capability to suggest alternative routes with the same cost.

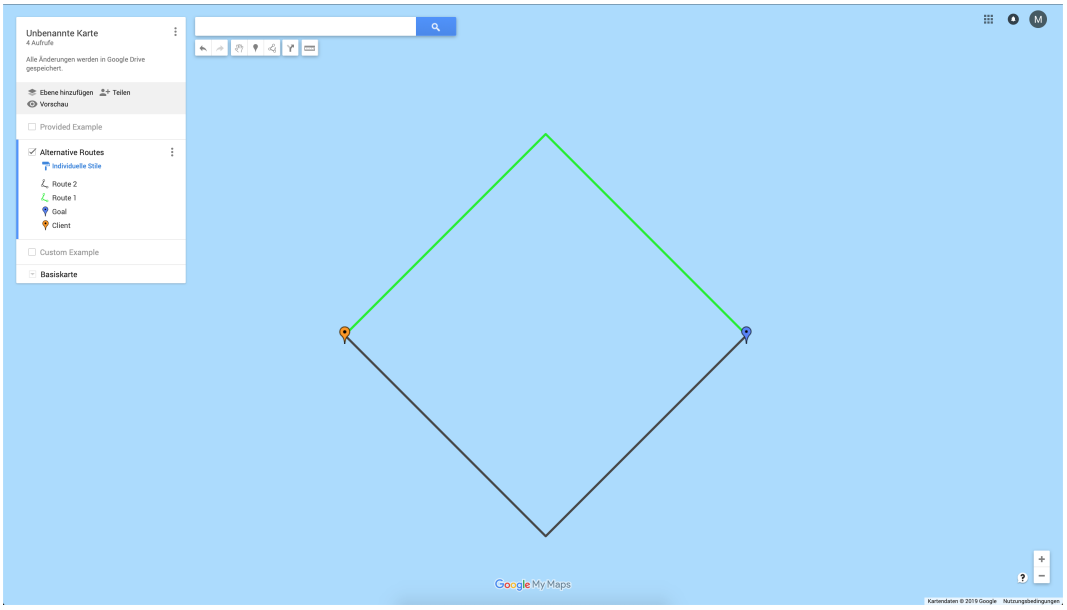

Figure 5: Custom map

4 Used external libraries

Name	Usage
Google Guava[1]:	Graph and HashMap data structures
Simon D.Levy's KDTree[2]:	Finding nearest neigbours to start and goal nodes
JDom2[3]:	Generating KML files

[1]https://github.com/google/guava
[2]https://home.wlu.edu/ levys/software/kd/
[3]http://www.jdom.org/