ENPM 661 Project 5: Highway Hierarchies

Rene Jacques and Zachary Zimits

Abstract—This paper is an attempt to implement Highway Hierarchy optimization on a Dijkstra search algorithm. Two different methods of neighborhood generation were attempted with the authors choosing to go forward with the method that defined neighborhoods based on geometry instead of the amount of nodes like in previous papers. While this method did not return the optimal path it was able to decrease the calculation time 20 times. The testing for this project utilizes Open Street Map (OSM) data from around the Washington DC area (DMV). This data was initially accessed and processed with the OSMnx python library.

Keywords—planning, highway hierarchy, Open Source Map, OSM, road network, graph, route planning, dijkstra, neighborhoods

I. INTRODUCTION

There are many levels of planning used when creating autonomous vehicles, from the trajectory of the vehicle in its lane to the higher-level planning from the departure point (A) to the destination point (B). This higher-level planning will be referred to as route planning for the remainder of this paper. In route planning nodes represent intersections and the paths between them are edges. This can pose a problem for certain search algorithms. When trying to find the path between two points in a road network the search complexity can expand exponentially with the distance between the two points when using Dijkstra algorithm. to combat this issue several techniques have been developed to simplify the planning problem and to speed up computation times. [3] One such algorithm is Highway Hierarchies (HH). The purpose of the Highway Hierarchy optimization is to recompute certain portions of the map so they can be generalized as a single node. This way the algorithm only has to do a classical heuristic search in close proximity to the start and end nodes. [1]

II. METHOD

A. Highway Hierarchy Structure

Highway hierarchies serve to abstract a map composed of nodes and edges into a smaller map with less nodes where each nodes represents a set of nodes from the original map. As many new maps as are needed or desired can be created to abstract the map further, resulting in several layers of maps that can be searched to find optimal paths. Each of these layers has less nodes than the layer preceding it, meaning that higher level maps can be searched faster than any lower level map.

The edges between each node in each map are composed of the roads that form the shortest path with the least cost between the entrance points of each node. This is shown in (1) where the path between the small red and blue nodes represents the edges between the larger nodes (shown as the shaded red and blue regions). Two nodes are connected

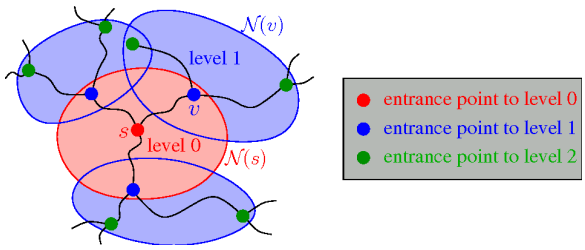

Fig. 1. Representation of HH search process. [6]

through their shared edge by entrances (sometimes refereed to as exits the terms are interchangeable), which are nodes from the original map that link the roads that form the shared edge. The larger nodes may have any number of entrances. The larger nodes also contain the shortest paths between each entrance and the other entrances within the node, enabling fast pathing once any search has been completed.

B. OpenStreetMap Map Processing

The data used to construct the base map is obtained from OpenStreetMap (OSM), an open set of map data that is community built. The map data can be accessed directly from the OSM website or by using their API. The python library we used to access the OSM data is OSMnx[7], a custom module designed to make accessing and plotting OSM data easy and efficient. OSMnx allows the user to access map data through either longitude and latitude or by using street addresses and place names. We specified a bounding box defined by the two points in table I to obtain (2) from OSM using OSMnx. The nodes and edges as

TABLE I
BOUNDING BOX POINTS

Longitude	Latitude (deg)
39.05	-76.96
38.92	-77.05

defined by OSMnx were then extracted in order to create a

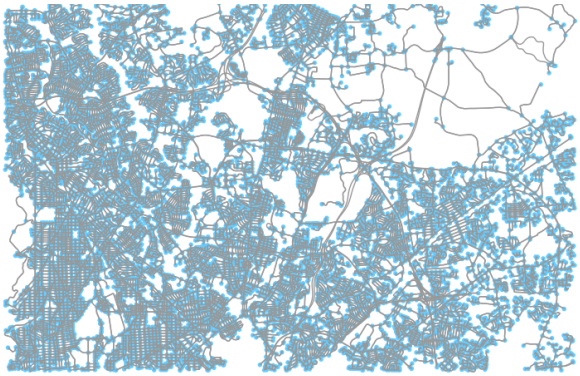

Fig. 2. Representation of HH search process. [6]

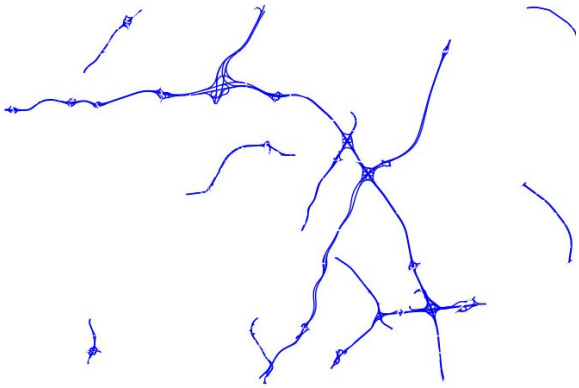

Fig. 3. Level 3 Roads: Motorways and Trunk Roads

base map (graph) that can be used to form the HH nodes.

OSMnx edges contain two dictionaries containing the information about each edge. These dictionaries are called keys, and data. Data contains the road type for each edge, which is used to sort each edge by type to enable simplified hierarchy searching within the base map. For example, the first type of road to be searched might be residential, and then primary, and then finally motorway. With each progressive step the search algorithm would only examine roads that matched the current road hierarchy, and therefore would only consider roads with a certain speed limit, making the path that is being followed faster as the levels are increased and slower as the levels are decreased. This allows for efficient map searching at level 0 (the base map).

TABLE II
OSM ROAD TYPES

Type	Level	Speed
motorway	3	65
trunk	3	65
primary	2	45
secondary	1	35
tertiary	1	35
residential	0	25
unclassified	0	25

$$cost = \frac{\text{edge length}}{speed} \tag{1}$$

Table II shows the road types as defined by OSM. The level column indicates the hierarchy that we have assigned to each type. Motorways cover major multi-lane roads and major divided highways while trunk roads cover the most important roads in a country that are not motorways, and are not necessarily divided. Motorways and trunk road are both considered to be level 1 since they are very similar in their usage in the United States. Trunk roads may differ more noticeably in other countries but for now they are considered to be the same as motorways. Figure 2 with only the motorways and trunk roads is shown in (3).

Primary roads are classified as the next most important roads after trunk roads, and are often found linking large towns together. Figure 2 with only the primary roads is shown in (4).

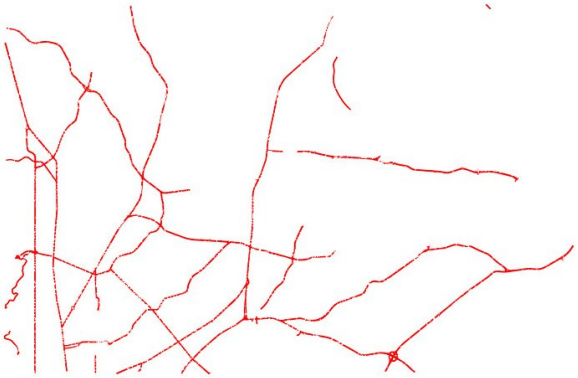

Fig. 4. Level 2 Roads: Primary Roads

Secondary roads covers the next most important roads after primary roads, and are often the linking roads between average sized towns. Tertiary roads are the next most important roads after secondary roads, and often link small towns and villages. Due to this similarity secondary and tertiary roads are considered to be on the same level. Figure 2 with only the secondary and tertiary roads is shown in (5).

Fig. 5. Level 1 Roads: Secondary and Tertiary

Residential roads are any road that accesses a residential property. For example this would include neighborhood roads which are lined with houses. This is the most common type of road in the base map. Figure 2 with only the residential roads is shown in (6).

Fig. 6. Level 0 Roads: Residential Roads

Finally, unclassified roads are classified as any road that is of lower importance than tertiary roads, serves some purpose, and does not access residential property. There are very few of these roads in the base map. Figure 2 with only the residential roads is shown in (7).

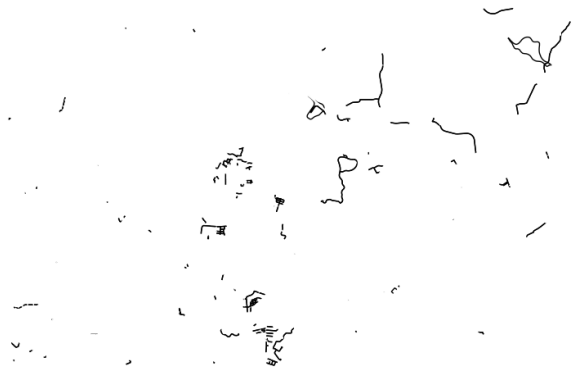

Fig. 7. Level -1 Roads: Unclassified Roads

All of these color coded road classifications combined into one map can be seen in (8).

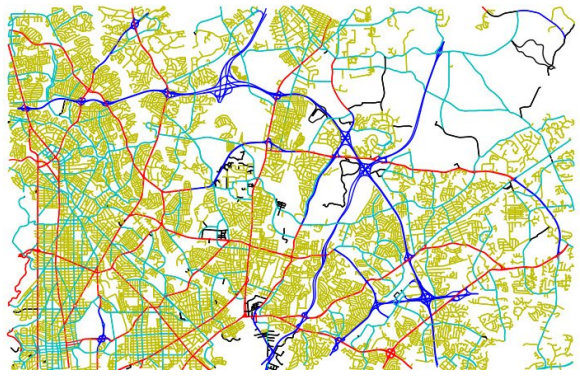

Fig. 8. Level -1 Roads: Unclassified Roads

With all of the road types extracted and sorted based on assigned level, the data describing each node and edge can now be stored in a manner that is useful for searching through. To do this we created a node class where each node object contains its coordinates and a list of all of its edges. We also created an edge class where each edge object contains: its type, the speed limit, its two nodes, its length. Iterating through all of the nodes and edges returned by OSMnx we were then able to create a graph of node and edge objects that represents our base map.

C. Graph Generation

The papers we looked at did not go into detail on how to create the neighborhoods only that they were defined by the number of nodes inside of them (H). Our initial thought was to use Dijkstra to create the neighborhoods. We started at a random point and ran a Dijkstra algorithm until the total number of nodes in the open and closed set equaled H. The algorithm would then return both sets. The closed set would be added to a global closed set and the open set would be added to a global priority queue. The first node from the priority queue is taken as the new start node. Dijkstra is run on this node with the global closed set being passed in as the closed set for that node to ensure that the new search did not find any nodes that belonged to the previous neighborhood. When the algorithm found H nodes for the new neighborhood it would return the updated closed set and its open set. We then compared that open-set to the global open-set. We knew that any repeats could no longer start new neighborhoods so we removed them from the priority queue and treated them as possible entrances for optimal paths between neighborhoods. We then took the next node from the priority queue and repeated the process until every node had been processed.

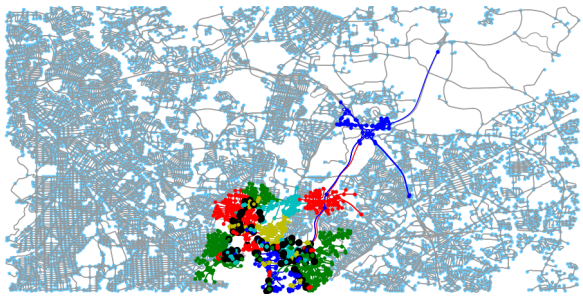

Fig. 9. Partial Results of Dijkstra Neighborhood Generation

This method did not quite work out, we found that by limiting Dijkstra to a certain size left a lot of instances where off-shoot roads or dead-ends would be missed by the first pass-over of Dijkstra and then another neighborhood creation algorithm would comeback and try to make it into a neighborhood but since every node surrounding it was already in a closed set the neighborhoods would only

consist of a few nodes. These instances could possibly be fixed with some post processing but for the purposes of this project we chose to focus our time on a different aspect and try another method of neighborhood generation. The partial results for this attempt can be seen in Figure 9. Individual neighborhoods are shown in alternating colors. In certain sections one can see an accumulation of large dots that signify start locations. These are the concentrations of small neighborhoods are keeping this method from working properly.

Our second method of finding neighborhoods is by plotting an array of hexagons over the map. We then use half planes to fit nodes to particular hexagons. To find the entrances we calculate which point is closest to each border and save that as a possible entrance. The issue with this method of finding entrances is that the entrance from one node my not be on the same edge as the entrance from the neighboring node. To fix this we reprocess every entrance and compare it to the corresponding entrance in the neighboring neighborhood. Which ever node is the closest to the border becomes the entrance for both nodes.

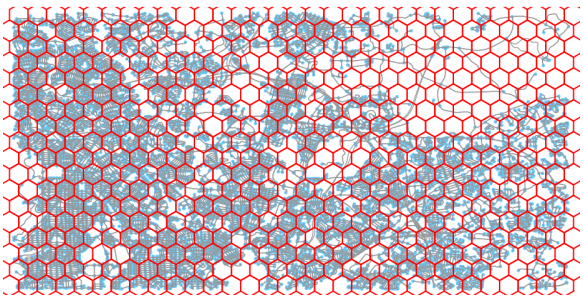

Fig. 10. Hexagon Overlay

Once all the neighborhoods have been defined we begin pre-computing the Dijkstra algorithm inside of each neighborhood. For each entrance we calculate the optimal path to the remaining five entrances for a total of 15 searches on each neighborhood. See Figure 11 and 12. These paths become our actions for the final Dijkstra that will be run later. Each path is saved with the list of nodes it used to get there and the total cost to get there. Due to the limited size of our data we are only implementing a single layer of neighborhoods. In order to implement multiple layers of neighborhoods a new neighborhood class would have to be made or the current one reworked in order to access the necessary data from the existing neighborhoods.

After all of the pre-processing is run each instance of a class is saved using the pickle python library. Each class has its own saved file that is loaded by the online planning algorithm so it can quickly access the list of nodes, edges, and neighborhoods.

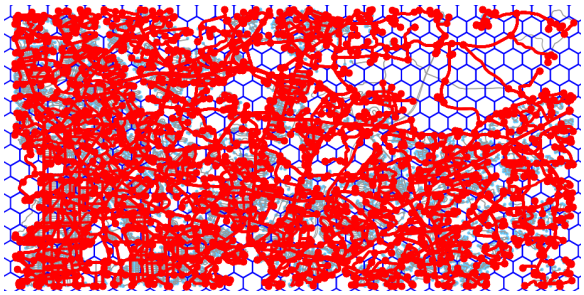

Fig. 11. Neighborhood Paths Pre-Computation

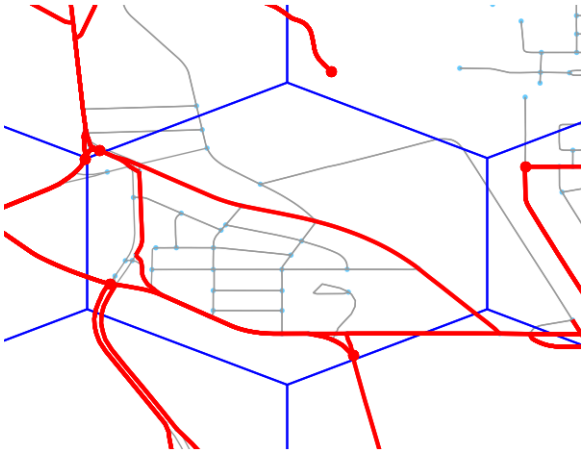

Fig. 12. Single Neighborhood Paths Pre-Computation

III. Implementation Using Dijkstra Algorithm

The planning algorithm for this begins like a normal Dijkstra search. Starting at the goal node it calculates the cost along the connecting edges to the next nodes and adds them to the open set corresponding the their cost. It then selects the first element of the open set and repeats the process. This continues until the algorithm finds one of the six entrances. This step can bee seen in Figure 13 where the first stage path is in cyan and the entrances are green dots.

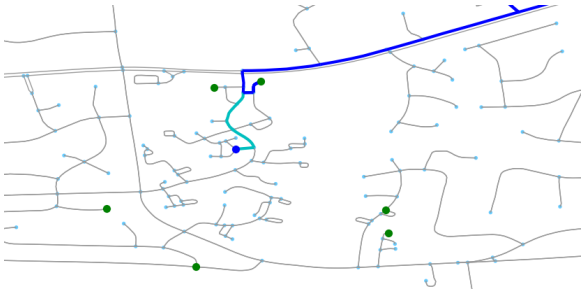

Fig. 13. Initial Dijkstra

Once at an entrance the next actions are defined by the possible paths from that given node. These paths are simplified into a single edge and the open set is updated with the the neighboring neighborhoods instead of individual nodes. This process is continued until the neighborhood is found that contains the goal node. The algorithm then reverts back to

a standard Dijkstra algorithm until the goal node is finally found. This portion of the algorithm is shown in blue in Figure 13.

IV. Previous Studies

Highway Hierarchies were first suggested in a paper by Peter Sanders and Dominik Schultes. [1] They put forward the idea that in order to speed up an optimal search algorithm such as Dijkstra requires either a guiding principle such as heuristic cost or to recompute some of the data. The first option using heuristic cost does not guarantee that the optimal path is returned. Alternatively when pre-computing data the developer must keep in mind the amount of memory that the stored data will take up. Their final algorithm was able to process the road network for the USA approximately 20 million modes in a few hours and could make searches in eight ms. This is approximately 2000 times faster than the standard Dijkstra algorithm.

In a later paper [6] Sanders and Dominik provide a revised version of the HH algorithm that can speed up query and pre-processing times. They were able to reduce the pre-processing time for 18 million nodes to 15 minutes and store the data as 68 bytes per node. With this setup the average search time is 0.76 ms.

In a later paper Schultes goes into more detail about the setup of the algorithm. The first step is to define the first layers of neighborhoods. These neighborhoods have a fixed size that can be tuned. [2] The shortest distance between nodes is calculated these are saved as new edges and all leafs that come off these main branches are ignored until the search reaches the goal nodes neighborhood to form our graph G_1. [4] If the search area is large enough the graph can be abstracted further by drawing new neighborhoods in G_1. We then make a similar simplification by only looking at the shortest paths between the neighborhoods. Again we ignore and branches that are not shortest paths. These steps create graph G_2 and can be repeated until G_n where n is a tune-able parameter. This decreases the complexity of the graph and makes the construction process faster for future iterations. The core of the highway can then be processed further to construct lower levels of highway graphs.

The search will begin with the start node in the original graph G_l. A local search of the start nodes neighborhood is performed until an edge leaves the neighborhood. At this point it changes to the next highest search level where the node that attaches to the edge that left the neighborhood is the entrance point into the neighborhood. In this higher level we can now continue to search normally until we reach the goal, switching levels as needed.

For the data for this project we are using the Open Street Map data so we needed a way to process the data and get out only what we needed. We found the python library OSMnx written by Geoff Boeing who outlines his library in an article posted in Computers Environment and Urban Systems [7]. Road networks can be represented as graphs, which are the representation of nodes and the edges between them. Undirected graphs like those used in [1] if the edges connecting nodes point in both directions. OSM uses multidigraph meaning that nodes can have parallel edges between nodes as well as having directed edges. They are also non-planar meaning that the network can not be simplified to 2D and the edges are weighted.

V. Results

Before running our HH algorithm we tested our map with regular Dijkstra and found the map shown in Figure 15. This map looks exactly how one would expect. The path finds its way to a highway and travels along it for the majority of the way due to its high efficiency. When we ran our algorithm on the same start and goal nodes and found the path shown in Figure 14. When comparing the two it is obvious that our calculated path is far from optimal.

TABLE III
Results of differnt Algorithms

Test Case	Dijkstra (sec)	Highway Hierarchy (sec)	Ratio
1	4.445	0.225	1:19.8
2	4.238	0.178	1:23.8
3	2.600	0.090	1:28.9
4	3.307	0.097	1:34.1
5	4.065	0.132	1:30.8
6	0.762	0.071	1:10.7
7	3.340	0.134	1:24.9
8	5.010	0.168	1:29.8
9	4.369	0.077	1:56.7
10	0.731	0.065	1:11.3

TABLE IV
Test Case Information

Test Case	Start Position (deg)	Goal Position (deg)
1	49171576	50651343
2	49811183	50656963
3	50445628	50543902
4	50527139	49766358
5	49163552	50297490
6	1563065123	50515020
7	49819801	50229553
8	49806714	50556577
9	50315348	50514418
10	50742515	50493250

Due to the way that we built our neighborhoods the entrances and exits do not necessarily lie on the optimal path between neighborhoods. As can be seen in Figure 16 the entrances forced the path to exit the highway and reenter it again.

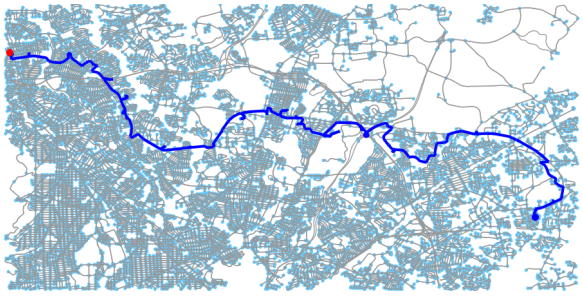

Fig. 14. Results from Highway Hierarchy

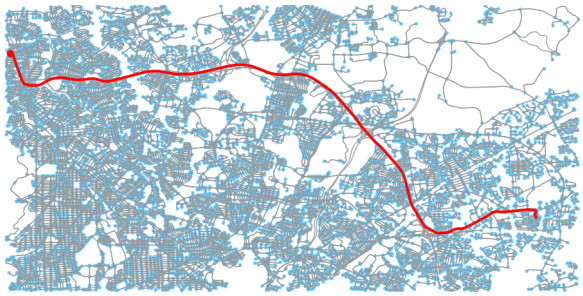

Fig. 15. Result from Standard Dijkstra

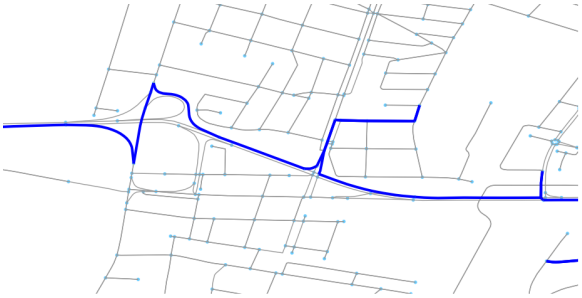

Fig. 16. Zoomed In On Error Spot

While the path is not optimal, which we expected, we did find that the HH method drastically sped up the search. The results from all 10 test cases we ran can be seen in Table III and IV For most of the tests the HH optimization was able to improve the standard Dijkstra search by over 20 times. In the cases where there was less improvement it is when the start and goal nodes are closer together where Dijkstra does not get as slowed down by an increasing node count.

VI. Conclusion

In conclusion, our results show that the highway hierarchies pre-processing technique can result in significant search time improvements over unmodified search algorithms. When paired with the Dijkstra algorithm, HH resulted in an average of approximately 27 times faster search speeds when compared to Dijkstra without HH. This improvement is a farcry from the 2000 times found in our papers [6]. They were running HH on the entire United States. If we were to expand our data set to include cross-country routes with multi-layer neighborhoods it is possible that we could see the same improvement. The trends indicated by our data suggest that more nodes result in greater differences. The path found by our implementation of HH is not optimal, especially when compared with unmodified Dijkstra, This could be improved without affecting search times if the optimal paths between neighborhoods were fully computed. While we are not exactly sure how the best method to do that is we believe that we were on the right track with our initial attempt of using Dijkstra to calculate the neighborhoods. If we added in enough checks to stop it from looping over small neighborhoods it might be a feasible method of generating neighborhoods. Even without finding the optimal path, our implementation of HH clearly shows that it is a fast and viable solution to map path planning search times.

APPENDIX

A. *Test Cases*

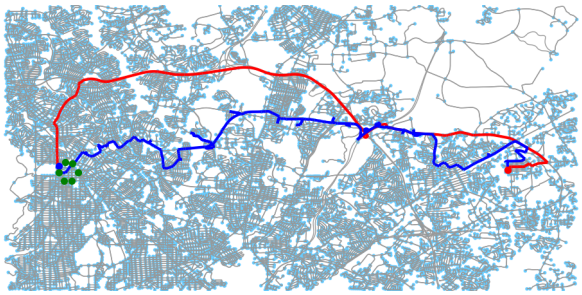

Fig. 17. Test Case 2

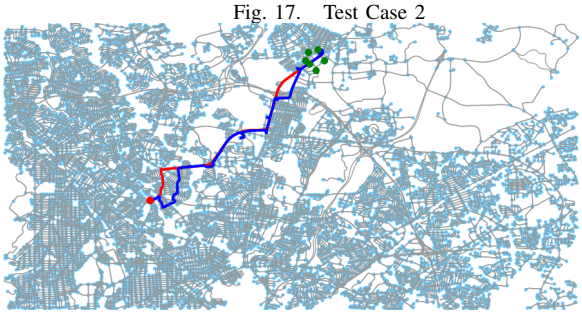

Fig. 18. Test Case 3

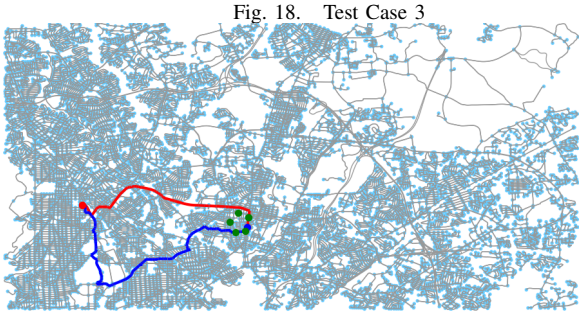

Fig. 19. Test Case 4

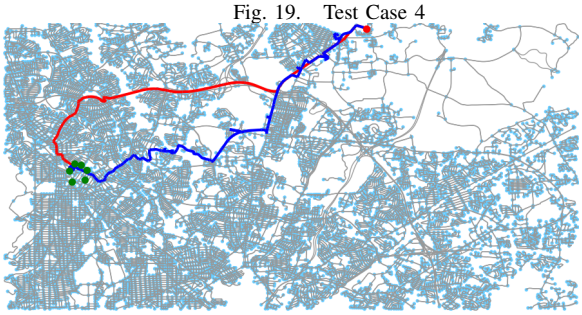

Fig. 20. Test Case 5

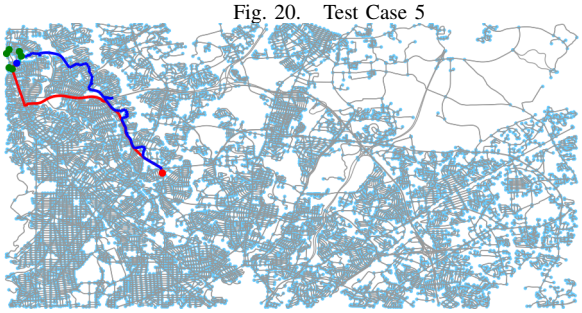

Fig. 21. Test Case 6

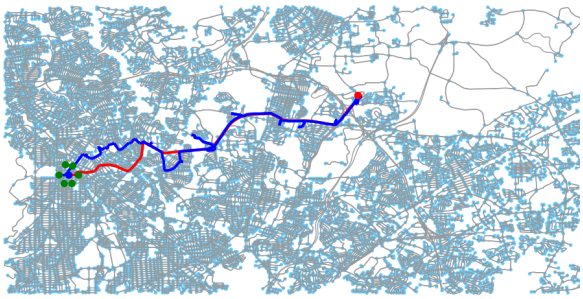

Fig. 22. Test Case 7

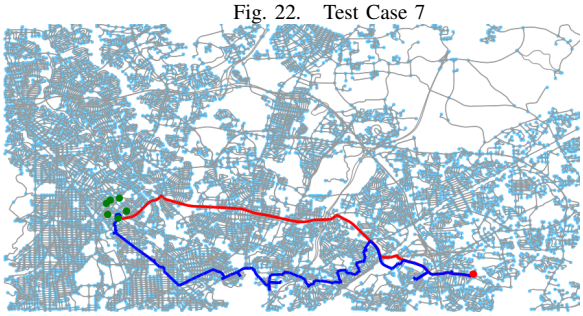

Fig. 23. Test Case 8

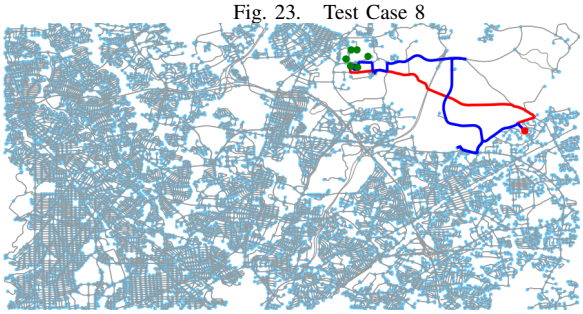

Fig. 24. Test Case 9

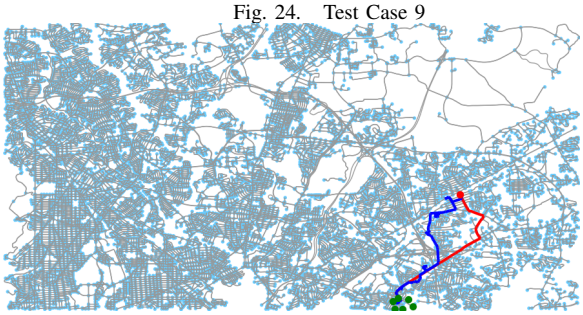

Fig. 25. Test Case 10

REFERENCES

[1] Sanders P., Schultes D. (2005) Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal G.S., Leonardi S. (eds) Algorithms ESA 2005. ESA 2005. Lecture Notes in Computer Science, vol 3669. Springer, Berlin, Heidelberg

[2] Sanders P., Schultes D. (2006) Engineering Highway Hierarchies. In: Azar Y., Erlebach T. (eds) Algorithms ESA 2006. ESA 2006. Lecture Notes in Computer Science, vol 4168. Springer, Berlin, Heidelberg

[3] Sanders P., Schultes D. (2007) Engineering Fast Route Planning Algorithms. In : Demetrescu C. (eds) Experimental Algorithms. WEA 2007. Lecture Notes in Computer Science, vol 4525. Springer, Berlin, Heidelberg

[4] Dominik Schultes, 2008, Route Planning in Road Networks, Ph.D. Thesis

[5] Fan D., Shi P. (2010) Improvement of Dijkstras algorithm and its application in route planning, Seventh International Conference on Fuzzy Systems and Knowledge Discovery, Yantai, China 2010. Yantai, China, IEEE

[6] Delling, Daniel, Peter Sanders, Dominik Schultes and Dorothea Wagner. Highway Hierarchies Star. The Shortest Path Problem, 2006.

[7] Boeing, Geoff. (2017). OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. Computers Environment and Urban Systems.

[8] E. Dijkstra. (1959). A note on two problems in connexion with graphs. Numerische Mathematik

[9] Goldberg, A.V., Harrelson, C. (2005). Computing the shortest path: A search meets graph theory. SODA.

[10] Willhalm, T. (2005). Engineering shortest paths and layout algorithms for large graphs.