A* Search on a Real Road Network with a Modified Heuristic

Sam Kleist
kleis043@umn.edu

May 11, 2016

Abstract

The experiment uses a modified heuristic for A* search on a real road network. This road network was the city of Minneapolis. The original heuristic for this problem calculated the distance traveled plus the straight line distance to the goal. It was found that a modified heuristic which considers left and right turns can greatly reduce the number of turns taken with only a slight increase in total distance. Other research on this topic was discussed and considered in the design of the experiment. Future modifications of this work would include testing the effects of other heuristics. These other heuristics may include traffic, weather and construction. Overall, it was learned that a modified heuristic on A* search can provide one with valuable results.

1 Introduction

Route Finding is a topic that has become much more prevalent as technology and mobility has increased. In computer science, route finding is simply the problem of finding paths in an environment. These environments include graphs, maps, and even the real world. A common route finding problem is the problem of the shortest path. As people have become more reliant on navigation systems, it is important that they are receiving information that is accurate and optimal. The shortest path problem is an interesting problem since there are many ways to solve it and also many ways for it be very complex. Todays GPS navigation systems must account for numerous factors when computing the shortest path. Although distance plays a large part in the computation, traffic, speed limits, construction and other factors must be taken into consideration as well. All these variables quickly make designing algorithms to solve the shortest path much more complex. There are many different algorithms that can be used to solve the shortest path problem. Some brute force algorithms that can be used are depth first search and breadth first search. Although these types of search algorithms are simple to understand and implement, they have a high time and space complexity which makes them only efficient on small graphs. These algorithms would be insufficient for large graphs such as a road network of the United States. By adding heuristics we are able to search graphs more intelligently and efficiently. A common directed search algorithm is A* Search. A* search uses an admissible and consistent heuristic to help guide its search. With an appropriate heuristic, A* search is optimal and much faster than depth first and breadth first search.

The problem I will be addressing in this paper is route finding on a real road network. I will be constructing my own A* search and testing it on the streets of Minneapolis. I will also be modifying the heuristic to include a cost for left and right turns. Since left and right turns often

force a driver to slow down or stop, they will be included in the cost of the shortest path. The effect that this has on the search will be observed and analyze. I have chosen A* search since it is a greedy algorithm that is well suited for the task at hand. The problem of route finding with a modified heuristic is an interesting problem as it is important to understand the other factors in finding the shortest path other than distance. When traveling, the fastest path is more of a concern than the shortest path. Since the Minneapolis data I will be testing on only includes coordinates and street directions, I will only be able to implement the heuristic for left and right turns. If presented with more data in the future, I would like to modify the heuristic even further and analyze the results. For example, a highway has a speed limit far greater than a city street. It is often desirable to travel on a highway when possible because of this faster speed limit. Other modifications to my A* search could include synchronization with real time data. Although speed limits are static, road conditions are constantly changing for many reasons. Some examples of dynamic factors are weather, traffic and accidents. In future versions of the A* search, the heuristic could include real time information about variables such as these when computing the cost of travel.

2 Literature Review

Route finding is an interesting problem that has been studied by many people in recent history. This research has helped further our understanding in many areas, such as how the brain functions and how we can make our daily lives easier. In 2004, the paper Route Finding by Rats in an Open Arena by Rebecca A. Reid and Alliston K. Reid [6] was published. The intent of this paper was to study the behavior of rats in an open arena when navigating to food. This paper helps to understand the heuristic a rat might use and what the different variables are forming that heuristic. These include, but are not limited to, minimizing energy, trail following, and taking the shortest path. The experiment was performed by placing a rat in an open arena with two food cups. The scientists then measured the path the rat took once it was placed in the cage. It was found that at first, the routes seemed random, but once the rats realized that there was food inside the food cups their paths became shorter and more predictable. Understanding, how rats behave and prioritize the paths they take can help humans prioritize their own routes as well. Another area looking into the topic of route finding is virtual reality. The paper, Navigation and Wayfinding in Virtual Reality: Finding the Proper Tools and Cues to Enhance Navigational Awareness by Glenna A Satalich [7] discusses how route finding can carry over into the virtual world. Satalich wonders if humans will interact with and recognize their surroundings in the virtual world as well as they do in the real world. The experiments performed show that if given a map, humans are able to navigate similarly to the way they would normally in the real world. Both rats and virtual reality provide interesting insight to the problem of route finding. As my focus is finding the shortest path on a real road network, it is important to consider how other specimens behave in a different environments and how that knowledge can be used to create your own heuristics. The following papers that will be discussed take a more in depth at the problem of route finding on real road networks. They discuss different heuristics and search algorithms that can be used.

The first paper I will discuss, Route Finding in Street Maps by Computers and People by R. J. Elliott and M. E. Lesk [2], introduces different ideas for prioritizing routes. The paper states that the shortest path may not always be the best path. This is because the shortest path often contains numerous turns. Elliot and Lesk add a cost to each turn, 1/8 mile for a right turn and 1/4 mile for a left turn. The search algorithms used are breadth first search and depth first search. They

implement each algorithm with single ended and double ended searches. Single ended searches search solely from the starting point to the destination. Double ended searches work from the starting point and the destination until they meet in the middle. It is found that depth first search is the fastest algorithm. However, the routes found by depth first search were slightly longer. Also, it is founded that double ended search finds better routes and is faster than single ended search. The next paper, Shortest Path Algorithms: An Evaluation using Real Road Networks by F. Benjamin Zhan and Charles E. Noon [8], discusses 15 different search algorithms to find the shortest path on real road networks. They claim that while much research has been done on finding the shortest path, the results are not reliable since the algorithms are often tested with randomly generated graphs. Randomly generated graphs contain irregularities that do not emulate real roads. After testing the 15 different algorithms on 10 different state maps, it is discovered that Dijkstras Approximate Buckets implementation is the best for finding the shortest path between two nodes. However, if you are generating a tree for all paths from a given node, TWO-Q and PAPE are the fastest algorithms. Zhan and Noon conclude by stating that the algorithms BF, BFP, and DKQ should not be used for the shortest path problem. The paper, Computing the Shortest Path: A* Search Meets Graph Theory by Andrew V. Goldberg and Chris Harrelson [3], discusses using A* search on the shortest path problem. Goldberg and Harrison explore many variants of A* and other search algorithms. The algorithms were tested on real road networks as well as randomly generated graphs. They introduce a preprocessing lower-bound method which computes shortest paths between landmarks before the actual search. They use the precomputed paths to help guide A* search. They title this modification of A* an ALT Algorithm. It is found that the ALT algorithm performs better than previous shortest path search algorithms if there is an adequate amount search space and precomputed landmarks. If this criteria is not met, Gutmans algorithm will perform better. Finally, the paper Finding the shortest route between two points in a network by Nicholson, T. Alastair J [5], discusses a way to find the shortest path in a graph. Similar to A* search, Nicholson proposes a greedy algorithm that only considers the surrounding nodes. Nicholson performs his search by always expanding the node that will lead to the lowest overall search cost. This will ensure that when the goal has been reached, the path found will be the shortest possible. Although this approach is optimal, its complexity can increase tremendously when dealing with dense networks.

These papers discussed each use different methods of solving the shortest path problem on road networks. Elliot and Lesk used breadth first search and depth first search to solve the shortest path problem. This was the only paper that used a brute force type search technique. Although the search technique was not great, Elliot and Lesk introduced interesting ideas for heuristics. They introduced the concept of associating a cost with turns. Also, they explored human algorithms. In an experiment, they asked real people to find the shortest path and used that human input to formulate their own ideas. This was the only paper that evaluated the thought process of humans. Elliot and Lesk introduced new valuable ideas to the problem, but used an approach that was simplistic and non-optimal. The ALT algorithm introduced in the second paper goes beyond left and right turns and considers costs such as current traffic patterns. Also, the second and third papers discuss algorithms that search more intelligently then the first paper. These two papers used searches that were guided by precomputed knowledge. The ALT algorithm used knowledge gained from preprocessing shortest paths between landmarks. Without the preprocessing, the ALT algorithm performs significantly worse. Also, the preprocessing done by the ALT algorithm is the most expensive computation of any of the three papers. The paper by Zhan and Noon show that

Dijkstras Approximate Bucket Implementation is best for finding the shortest path between nodes. The knowledge base of this algorithm consisted of real road networks that were sufficiently larger in size than the networks in the first paper. In conclusion, each paper provides algorithms that are useful in different scenarios. Double ended depth first search and Nicholsons algorithm are great if one is looking for a simple implementation on smaller road networks. However, if the road networks become large, the search time of each of those algorithms increases dramatically. In general, Dijkstras Approximate Buckets Algorithm is faster than double ended breadth first search for any map when searching between two nodes. However, it is a more novel approach and harder to implement. Finally, the ALT Algorithm may be the fastest and smartest algorithm studied, but it is by far the most complex. Also, due to its preprocessing, it can be computationally expensive to implement.

There are many ways to solve the shortest path problem on road networks. The papers discussed above have provided much insight to heuristics and search algorithms that can be used for a good solution. These papers include many ideas that can be included in other searches such as the A* search that I have developed. Along with these papers discussed, researchers have implemented other techniques to solve the problem of route finding. The paper, Route finding by neural nets by Bugmann, Guido, John G. Taylor, and M. Denham [1], discuss using neural networks to aid in finding the best path. The neural network is built with two layers. The lower layer stores location data and the upper layer holds information about the relationship between different nodes. Using this neural network design, they are able to effectively design a neural network that implements route finding. Another paper further expanding the problem of route finding is Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding by Liu, Bing [4]. This paper goes into detail about how to use case-based reasoning to help make decisions in a dense network. In conclusion, much research has been done in the area of route finding. Due to this expansive amount of research, we have been able to build better search algorithms that have made the lives of everyday humans easier.

3 Approach and Algorithm Details

To solve the problem of the shortest path on a real road network I decided to write a program in Java. As an input, the program reads the file map.txt which contains the connections of all the intersections on the road network of Minneapolis. Each line of map.txt contains a number 1 or 2, which specifies a one or two way street, x and y coordinates of the starting point, and x and y coordinates of the ending point. When the program reads the file map.txt, the data is stored into an instance of the class called Map. Map contains the x and y coordinates of the starting and end points in a data type called coordinate and the type of street in an integer variable named road type. Once the program has stored the map in a way that is useful, the user inputs their starting point and destination. If the user enters a coordinate that doesnt exist in the data set, the program will find the nearest coordinate. After the user has inputted their desired path, they must also specify the cost of left and right turns. This cost added to left and right turns is entered in the form of a distance. For this experiment, I will be inputting two different costs for left and right turns and measuring the results. After all input has been gathered, the algorithm can run.

To solve this shortest path problem I used A* search. I have designed a class called ASearch that contains all the necessary components to run A* search. In the constructor, ASearch loads an instance of Map. This is the map that A* search will use. ASearch contains a function called

getNeighbors() which returns all neighboring intersections that can be reached from a specific coordinate. This function is used to at the beginning of the search in the function run(). The function run() contains all the pieces of A* search and also outputs the results. It begins by creating two sets, fringe and explored. They are initially empty. Also, a node named current is initialized. Node is a data type that I have designed specifically for ASearch. It contains a set of coordinates, a parent node, and the cost of the distance traveled. To begin the search, current is set equal to the starting node that the user has specified. This node is then added to the set explored. The initial node has a cost of 0 and a parent of null. The rest of the algorithm is executed in a while loop that terminates when the current node has coordinates equal to the coordinates of destination. Therefore, if the starting node is equal to the destination the algorithm will terminate and return current with a cost of 0. The while loop begins by creating a list called currentNeighbors. This list contains the results of the function getNeighbors(). For each coordinate in currentNeighbors, a new node is created. The coordinate of the each new node is taken from the list currentNeighbors. The parent of each new node is set equal to the current node and the cost of the each new node is set equal to the distance already traveled plus the distance to the destination. Also, if the user has specified a cost for left and right turns, this is added to the cost of the node. In the program, a left and right turn is determined by calculating the angle between the new node, the current node, and the parent of the current node. If the angle is between 0 and 135 degrees it is measured as a left turn and if the angle is between 225 and 360 degrees it is measured as a right turn. If one of these conditions is met, the respective cost of that turn is added to the cost of the new node. Now that the algorithm has built new nodes for all of the neighbors of current, it must decide whether or not they should be added to the fringe set. First, the algorithm checks whether or not a node with matching coordinates is already in the fringe set. If it is found to be in the fringe set, it compares the cost of the node in the fringe with the new node that has been created. If the cost of the new node is less than the cost of the node in the fringe, the node in the fringe is modified to have the parent of the new node and the cost of the new node. However, if the cost of the new node is greater than the cost of the node already in the fringe, the new node is ignored and not added to the fringe. Next, if the algorithm was not able to find a node with matching coordinates in the fringe, it searches the set explored to see if that node has already been explored. If a node with matching coordinates is found in explored, the new node is added to the fringe only if it has a cost less than the node found in explored. At this stage of the algorithm, all neighboring nodes that should be searched have been added to the fringe. Now, the algorithm simply removes the node from the fringe with the lowest cost and sets current equal to that node. The new current node is also added to the set explored. The algorithm continues until the coordinates of the current node are equal to the destination coordinates or the fringe becomes empty. If the fringe becomes empty we are unable to travel to the destination from the starting point with the current map. This may occur due to the fact that there are one way streets and the map is discontinuous along its edges. If the algorithm is able to reach the destination coordinates, the algorithm returns the path by iterating through the parents of each node until it reaches the starting node. During this process, the algorithm also adds up the number of left and right turns.

Finally, the program will output the set of coordinates that should be traveled, the distance from the starting point to the destination and the number of left and right turns taken to reach the destination. Also, the algorithm will output an html file that can be opened in the browser to visually see the path. The maps generated in the html files will be used as visuals for my experiment results. This algorithm contains many loops that could be optimized in the future. However, with

the given map, all searches take less than a second to run so it is not extremely slow by any means. Now that I have laid out the basis of the algorithm and the program I have developed, I will lay out the experiment I have designed and the results received from that experiment.

4 Experiment Design and Results

For this experiment, I will use the Java program recently described to test the results of adding left and right turns to the heuristic of the shortest path problem. Similar to the paper referenced in the literature review section, I decided to use a cost of .5 miles for left turns and .25 miles for right turns. Since the map data provided was not in any measurable units, I used google maps to determine the distance between two coordinates. To do this, I measured the distance between two coordinates using google maps and the found those same coordinates the Minneapolis map. The maps compared and measured are shown below.

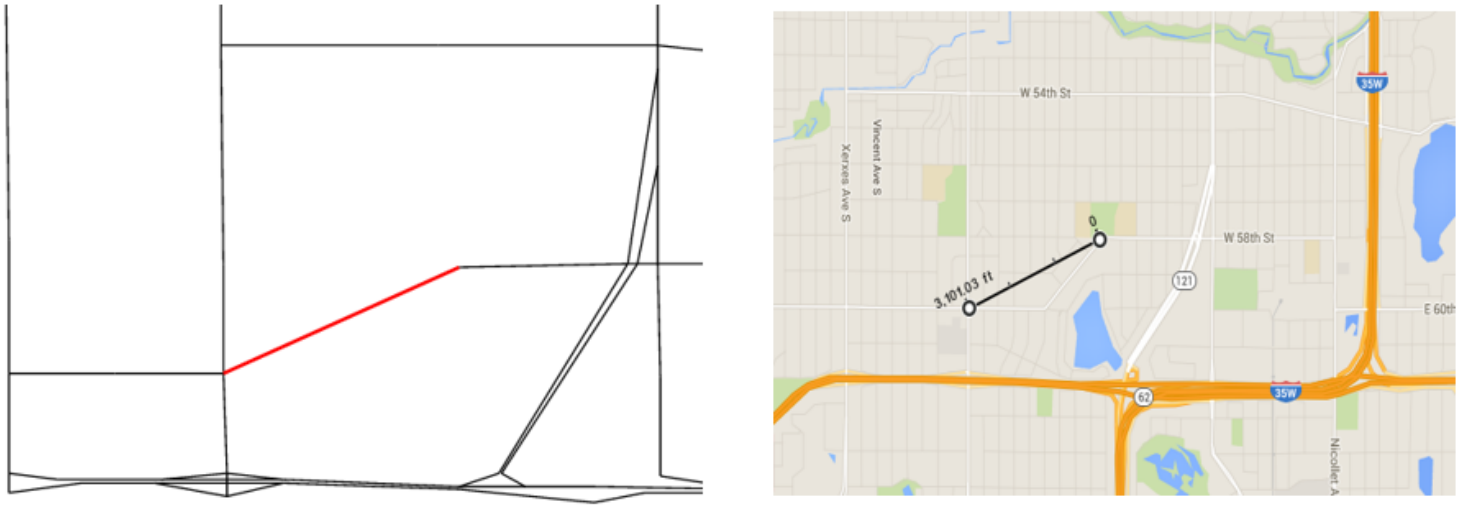
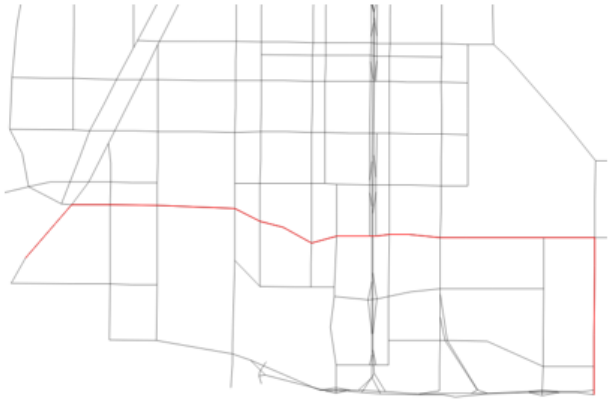


Figure 1: Maps compared to caluclate unit distance

From this test, I found that 1 unit was roughly equal to .5 meters. Therefore, 800 units was equal to a .5 miles and 400 units was equal to a .25 miles. In the program, 800 and 400 were added as the costs for left and right turns. Also, I ran tests with a cost of 0 for left and right turns. In the results, I was interested in seeing if there was any relationship between the distance added and turns removed. The ideal results of this experiment would be a large reduction in the number of left and right turns with a slight increase in distance to travel. To run my experiment, I will test 10 different starting and ending coordinates in the city of Minneapolis and measure the number of turns and the total distance for each path. Although this is a small sample size, it will be sufficient to understand the effects of the modified heuristics. Shown below are the 10 different paths tested. The maps shown on the left have no cost added to left and right turns and the maps on the right have a cost of .5 miles for left turns and .25 miles for right turns.

Distance: 2.31 miles

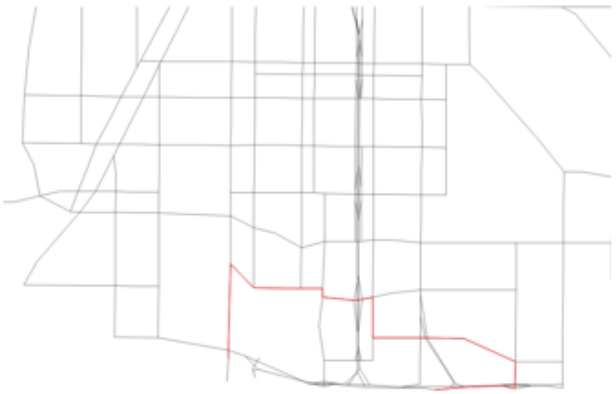
Left Turns: 2 Right Turns: 5



Distance: 2.36 miles

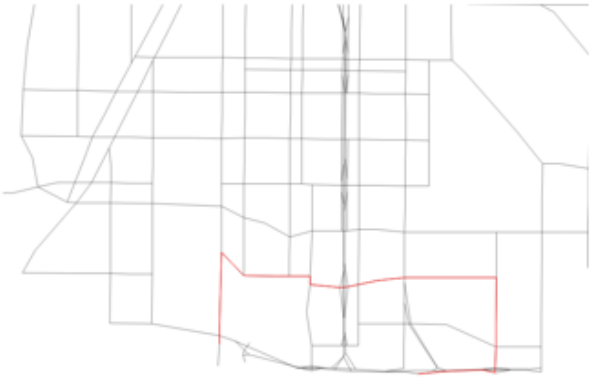
Left Turns: 0 Right Turns: 2

Figure 2: Trial 1



Distance: 1.86 miles

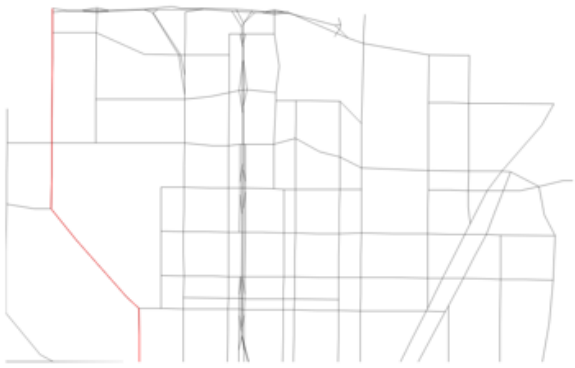
Left Turns: 2 Right Turns: 5



Distance: 1.95 miles

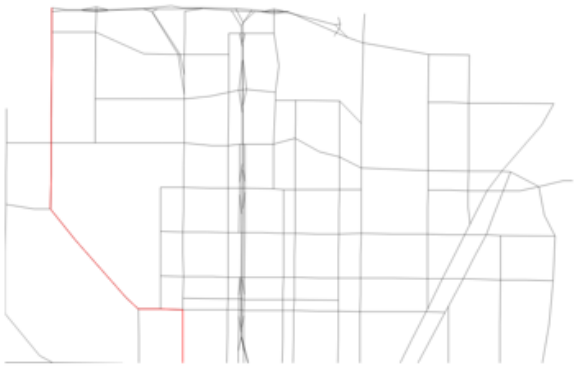
Left Turns: 1 Right Turns: 4

Figure 3: Trial 2



Distance: 3.97 miles

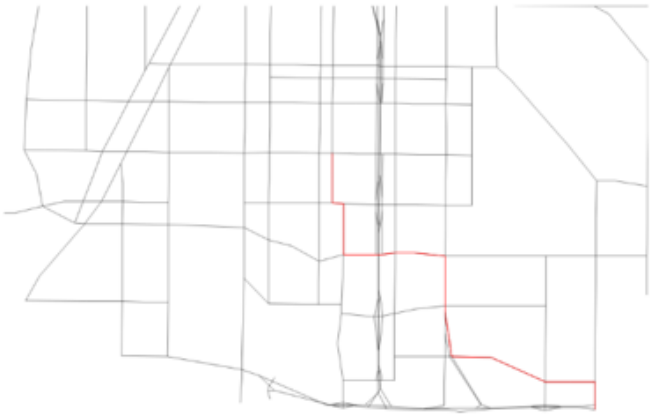
Left Turns: 7 Right Turns: 6



Distance: 4.22 miles

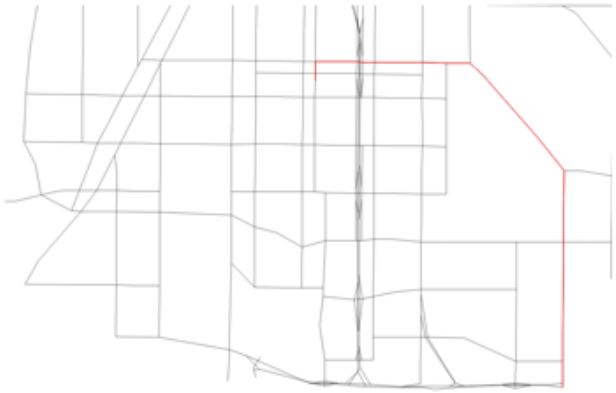
Left Turns: 3 Right Turns: 4

Figure 4: Trial 3



Distance: 1.66 miles

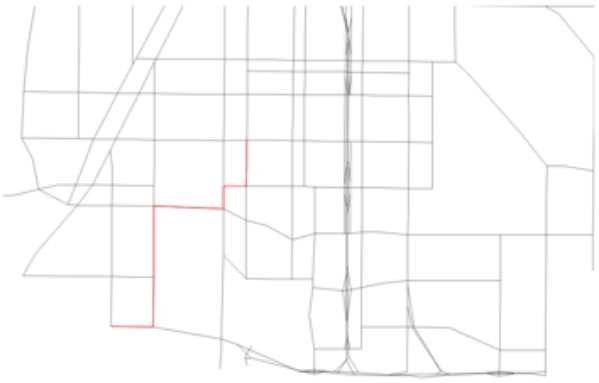
Left Turns: 3 Right Turns: 3



Distance: 1.78 miles

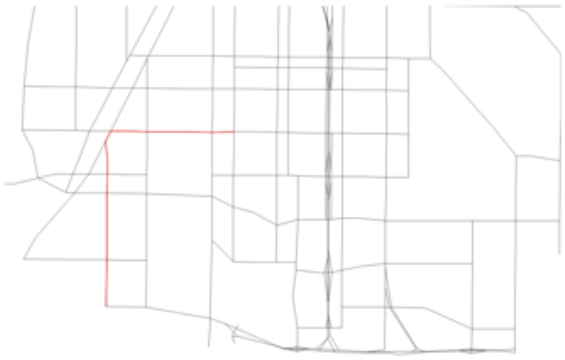
Left Turns: 0 Right Turns: 1

Figure 5: Trial 4



Distance: 1.16 miles

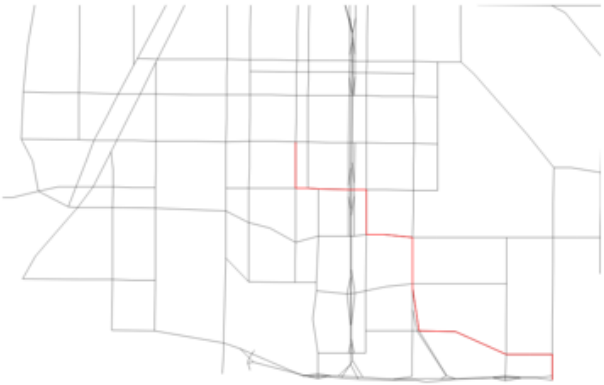
Left Turns: 3 Right Turns: 3



Distance: 1.2 miles

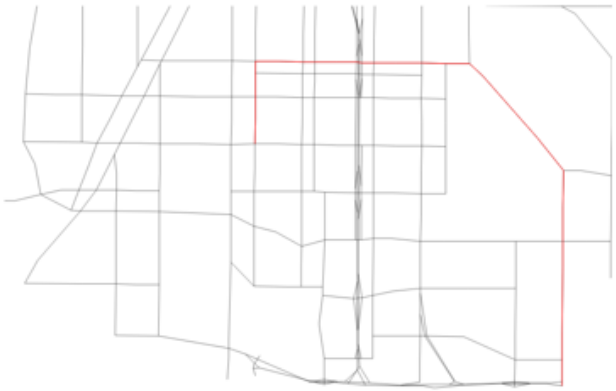
Left Turns: 1 Right Turns: 0

Figure 6: Trial 5



Distance: 1.61 miles

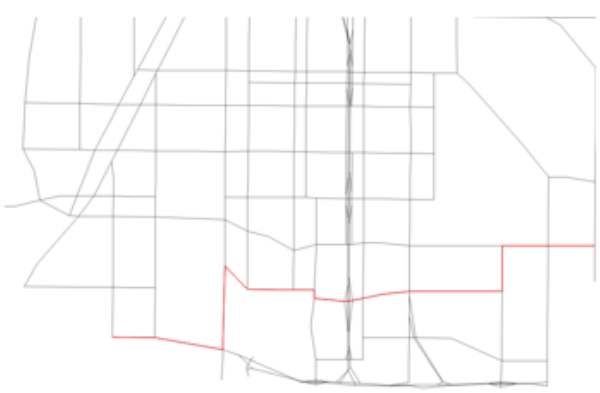
Left Turns: 3 Right Turns: 4



Distance: 2.2 miles

Left Turns: 0 Right Turns: 2

Figure 7: Trial 6



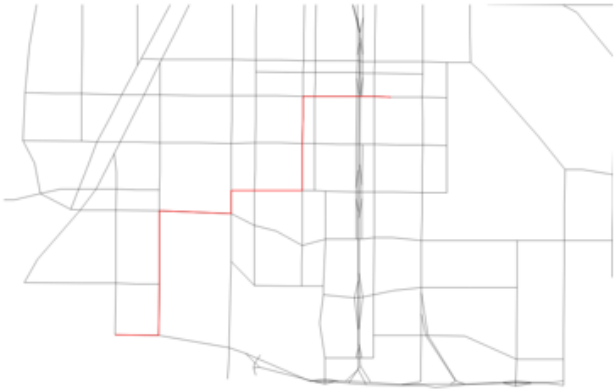
Distance: 2.21 miles

Left Turns: 3 Right Turns: 4

Distance: 2.42 miles

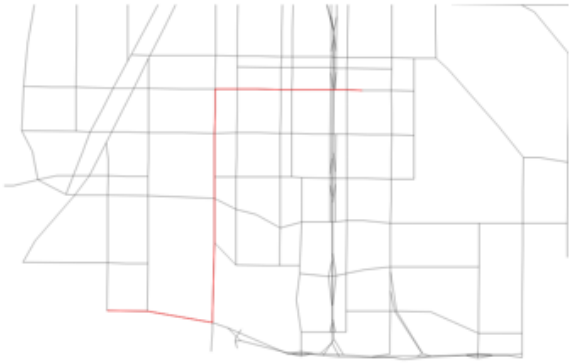
Left Turns: 1 Right Turns: 2

Figure 8: Trial 7



Distance: 1.81 miles

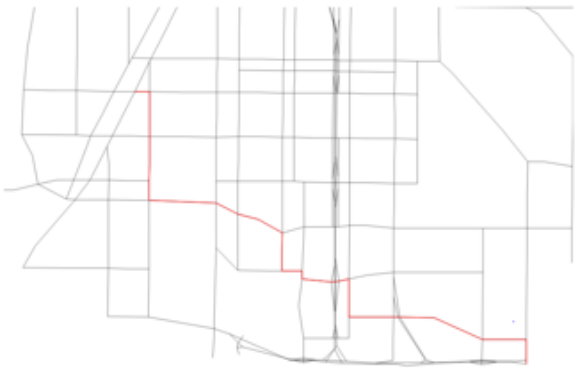
Left Turns: 3 Right Turns: 3



Distance: 2.06 miles

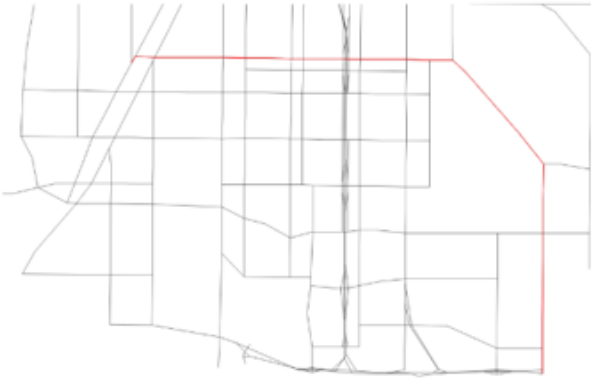
Left Turns: 1 Right Turns: 1

Figure 9: Trial 8



Distance: 2.23 miles

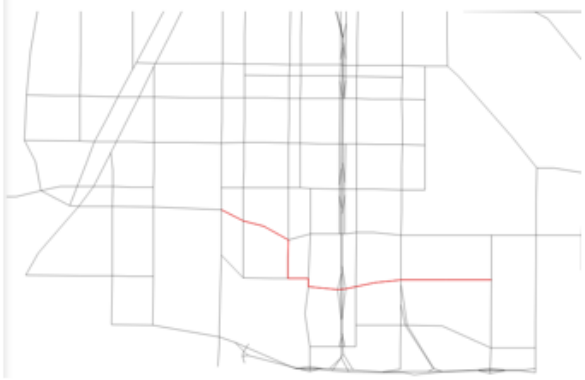
Left Turns: 4 Right Turns: 5



Distance: 2.36 miles

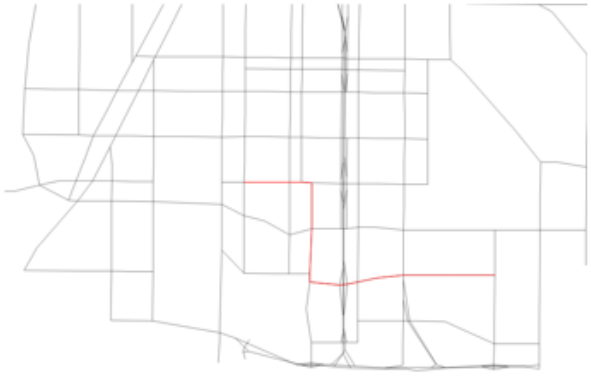
Left Turns: 0 Right Turns: 1

Figure 10: Trial 9



Distance: 1.19 miles

Left Turns: 3 Right Turns: 2



Distance: 1.27 miles

Left Turns: 1 Right Turns: 1

Figure 11: Trial 10

From the 10 trials, the modified heuristic added a combined total of 1.8 miles to the routes, but removed a total of 47 turns. Therefore, on average, we were able to remove 1 turn for every additional .05 miles. Also, we were able to remove a total of 25 left turns and 22 right turns. This is equal to removing 1 left turn for every additional .07 miles and 1 right turn for every additional .08 miles.

5 Analysis

The results found from the modified heuristic are promising. It has been found that A* search with a modified heuristic for left and right turns is in fact able to greatly reduce the number of turns taken. The uncertainty to this problem was the amount of distance that would be added. The results show that on average, an extra .05 miles is added to remove a turn. Also, .07 miles to remove a left turn and .08 miles to remove a right turn. The results favor the removal of left turns more than right turns due to the fact that left turns were given a cost of .5 a miles while right turns were given a cost of .25 miles. Therefore, one would expect the A* search to have a higher reduction of left than right turns. The modified heuristic has shown that it can be used effectively if it is important to reduce the number of turns. There are numerous factors that may cause someone to desire less turns. For example, in traffic, reducing the number of turns may save a great amount of time. The results can be analyzed by assessing how much time you would need to spend at a turn for it not to be worth your time to take the reduced turns path. Since we found that we were able to remove a turn with an extra .05 miles on average, we can assess how much time it may take to travel .05 miles. If the speed limit is 30 mph, .05 miles will take one approximately 6 seconds to travel .05 miles. Therefore, it can be argued that if you will spend more than 6 seconds at all turns compared to going to straight, it would be better to take the modified path.

It is important to remember that this problem may not always apply to all paths and there may be other factors which play a larger role for a commute. An example of this may be looking at speed limits. If the modified heuristic returns a path with less turns, but they are on much slower streets, it is likely not the optimal choice. Also, the algorithm does not account for dynamic factors such as weather and construction. To further test the program, I would try different costs for left and right turns. Also, I would analyze the results with left and right turns given the same cost. Although the heuristic we have provided has also been used in other research, there may be even better solutions. Also, with more data and more time, I would like to test A* search with other factors modifying the heuristic. For example, if the distance of a road was divided by the speed limit of that road, roads with higher speed limits would have lower costs than roads with lower speed limits of the same length. This could be easily tested with the gathering of more data. Also, real time weather data could be added into the heuristic. This would entail adding a set amount of cost per weather condition. For example, 0 miles could be added if an area was sunny, but .5 miles could be added if an area is rainy. A difficulty that one may run across using this heuristic is that weather only varies slightly in nearby locations. Also, if the weather heuristic was used for a cross country road trip, the weather may have change part way through the trip. To respond to these possible weather changes, the heuristic could also consider the forecast as well. There are numerous heuristic modifications that can be done and it would be interesting to add them into my A* search.

6 Conclusion

In conclusion, a modified heuristic for A* search on a real road network returns promising results. We were able to conclude that if one would be spending a great deal of time sitting at turns compared to going straight, it would be beneficial to use the modified which considers left and right turns. It was found that removing turns added on average .05 miles per turn. A greater cost was given to left turns since one usually spends more time taking a left turn versus a right turn. The experiment showed that a modified heuristic on the traditional A* search shortest path problem can provide valuable results applicable to the real world. In the future, I would like to test my Java program with even more heuristic modifications. These may include speed limits, weather, and traffic. Also, it may beneficial to test other search algorithms and measure difference in runtimes. A program such as the shortest path program should be as fast as possible in order to satisfy its users. However, A* search was a good choice due to its easily modifiable heuristic. Overall, it has been learned that one can greatly reduce the number of turns taken with only a minimal increase in total distance by using a modified heuristic on A* search.

7 Source Code

https://github.com/sam2k13/A-Search-on-a-Real-Road-Network-with-a-Modified-Heuristic

The url directs to a Githhub repository containing the source code I have written for this project. All Java code used can be found in the file Project.java.

References

[1] G. Bugmann, J. G. Taylor, and M. Denham. Route finding by neural nets. *Neural networks*, pages 217–230, 1995.

[2] R. Elliott and M. Lesk. Route finding in street maps by computers and people. In *AAAI*, pages 258–261, 1982.

[3] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[4] B. Liu, S.-H. Choo, S.-L. Lok, S.-M. Leong, S.-C. Lee, F.-P. Poon, and H.-H. Tan. Integrating case-based reasoning, knowledge-based approach and dijkstra algorithm for route finding. In *Artificial Intelligence for Applications, 1994., Proceedings of the Tenth Conference on*, pages 149–155. IEEE, 1994.

[5] T. A. J. Nicholson. Finding the shortest route between two points in a network. *The computer journal*, 9(3):275–280, 1966.

[6] R. A. Reid and A. K. Reid. Route finding by rats in an open arena. *Behavioural Processes*, 68(1):51–67, 2005.

[7] G. A. Satalich. *Navigation and wayfinding in virtual reality: Finding the proper tools and cues to enhance navigational awareness.* PhD thesis, University of Washington, 1995.

[8] F. B. Zhan and C. E. Noon. Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.