Engineering Route Planning Algorithms in Polar Coordinates

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Brian H. Nguyen

Spring, 2022.

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Upsorn Praphamontripong, Department of Computer Science

Engineering Route Planning Algorithms in Polar Coordinates

CS 4991 Capstone Report, 2022

Brian Nguyen
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
bhn5ger@virginia.edu

Abstract
Computing an optimal route in a network between a specified beginning and end destination is a frequent task when planning trips with public transportation and cars. Rigorous research has brought improvements to the functionality of classic route planning algorithms, but they have yet to be placed and examined in more complex settings. The objective of my technical research is to adapt existing route planning algorithms to more complex environments and, potentially, further optimize them. More specifically, I will place these algorithms in polar coordinates to gain insight as to how they work, which will be done by building a simulator in the form of a Java applet. Placing well-known route-planning algorithms in the simulator has already led to some remarkable observations, including an intuitive depiction of weighted versus unweighted algorithms using arc length. Although some progress has been made, going forward, more algorithms are to be implemented into the simulator, so I can convert observations into optimizations to make them more efficient and improve route planning in practice.

1 Introduction

There are many applications such as logistic planning that solve a large number of requests to calculate shortest-paths in transportation networks. Mobility is very important in our society; every time people travel from one location to another, they must determine the best route to reach the destination, accounting for random variables such as the day and week, time, congestion patterns, and construction. Car navigation systems are capable of taking over these tasks that are otherwise performed by the driver; using route optimization software leads to increased safety, reduced transportation expenses, and better planning. By having a clear route set out, the driver experiences a significant impact on risk reduction, potentially helping avoid accidents. Transportation expenses such as fuel and maintenance rise quickly from unplanned routes that result in long periods of driving time. For businesses that require scale to reach hundreds of destinations in one day, route planning arranges the destinations in a timely and efficient manner for the best solution. Optimal routing drastically cuts fuel costs, guarantees the safest path, and efficiently utilizes vehicles to allow businesses and individuals to achieve more.

2 Background

As described in the next section, there is a considerable amount of prior art related to the research topic, and one may argue that would leave little room for breakthroughs in the near future. However, most research in route planning algorithms is on *how they work*, not *how they are being used*. In other words, there has been rigorous research and improvements to the functionality of classic route planning algorithms, but they have yet to be placed and examined in more complex settings. In particular, route planning algorithms have only been commonly modeled on the Cartesian plane, using Euclidean distance. This research will initially challenge *how they are being used* by simulating them in more complex environments, particularly polar coordinates, to adapt them to such settings and then circle back to insights as to *how they work*.

3 Related Works

Currently, virtually all route planning algorithms use depth-first or breadth-first search as a basis [1]. In

fact, the most widely used algorithm for route planning is simply BFS weighted, Dijkstra's algorithm. The algorithm maintains the tentative distances for each node and settles the nodes of the road network in the order of their distance to the source node, until the target node is visited [2]. Bidirectional search is another classical route planning algorithm used today, which executes Dijkstra's simultaneously forward from the source and backwards from the destination [3]. These same algorithms are examined in this research, but in a non-Euclidean setting.

4 Project Design

Please see
https://github.com/bhn5ger/pathfinding-visualizer
for the full project, including source code and animations of the images that follow.

4.1 Procedure

To carry out the stated objective, the following methodology was performed:

1. Simulating existing route planning algorithms in polar coordinates
2. Making observations for potential to optimize

A Java applet was built from scratch for full control over simulation. The following sections discuss the applet's representation, components, and capabilities.

4.2 Weighted and Unweighted Graph Representations

The graph/grid is undirected, but either weighted or unweighted depending on the algorithm selected. Dijkstra's algorithm uses weights while depth-first search and breadth-first search do not. To determine the numerical weights of the weighted graph/grid for Dijkstra's algorithm, the measurements shown below in Figure 1 were used.

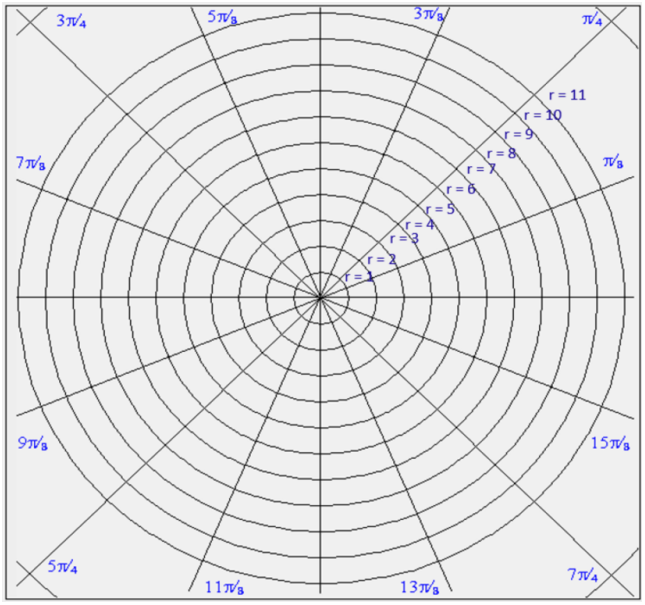

Figure 1: Polar Plane Measurements

With these radii and angles, weights for edges that stem left and right from nodes were calculated with the arc length formula $s = r\theta$, while weights for edges that stem forward and backward from nodes are 1 (because the distance from traveling forward and backward from nodes is constant). Figure 2 below is a weighted graph that roughly represents the polar grid for Dijkstra's.

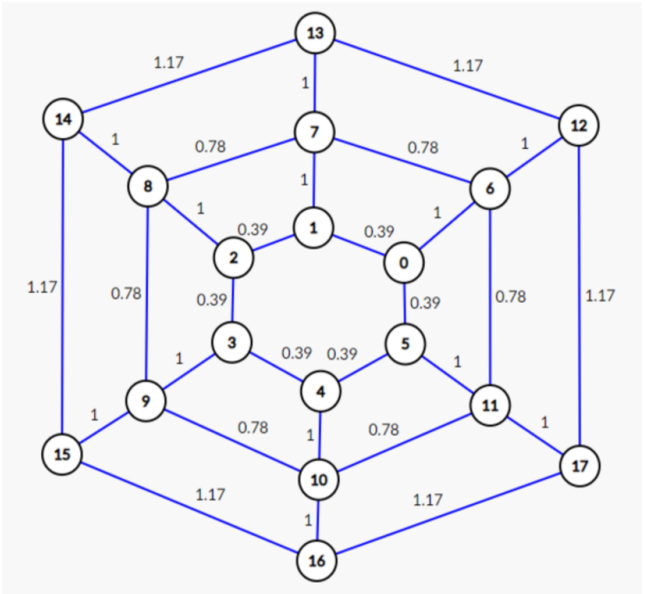

Figure 2: Weighted Graph Representation of Polar Grid

Note that this graph is not the exact one used in the application, and that the actual graph has 16 nodes per circle (instead of 6), 11 circles (instead of 3), and possibly obstructions. However, the weights between edges in the image properly depict that traveling left and right between nodes farther away from the origin is more costly (because of a greater arc length), whereas traveling forward and backward between nodes is the same distance anywhere on the graph. Consequently, Dijkstra's will choose the path that is closer to the center as going left and right is less costly when closer to the origin.

Since depth-first search and breadth-first search do not take weights into account, weights for all edges are 1. Figure 3 below is an unweighted graph that roughly represents the polar grid for these algorithms.

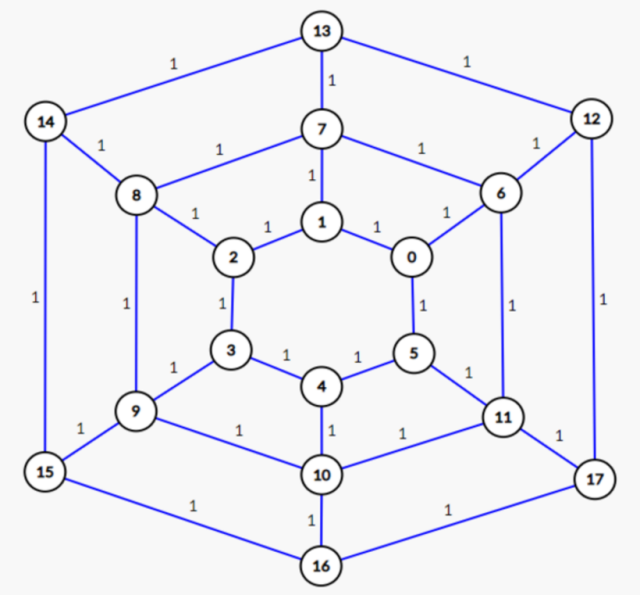

Figure 3: Unweighted Graph Representation of Polar Grid

Note again that this graph is not the exact one used in the application, and that the actual graph has 16 nodes per circle (instead of 6), 11 circles (instead of 3), and possibly obstructions.

4.3 Console and Control Panel

The console is located at the top left of the application and consists of a label to display instructions, information about algorithms, unweighted path and weighted path length. It also has a label to track the number of nodes checked.

The control panel is below the console and contains buttons to generate and clear maps, a toolbox for drawing a map, a drop down menu to select an algorithm, a slider to set animation speed, and finally a button to initiate searching. Clicking the clear map button resets the color of each node back to white, and clicking the generate map button creates a maze with a random start, finish, and random walls. The toolbox contains radio buttons to select between drawing a start (green), a finish (red), walls (black), and using an eraser (sets nodes back to white). The drop down menu allows you to pick from Dijkstra's algorithm, breadth-first search, and depth-first search, and updates the label that displays information about the algorithms in the console when selected. The slider controls the speed of the searching animation; the farther to the right the slider the quicker the animation. The search button initiates searching, but only works when there is a start and finish node marked on the grid and an algorithm selected. While searching, the control panel is disabled. Figure 4 below shows the console and control panel on the left.

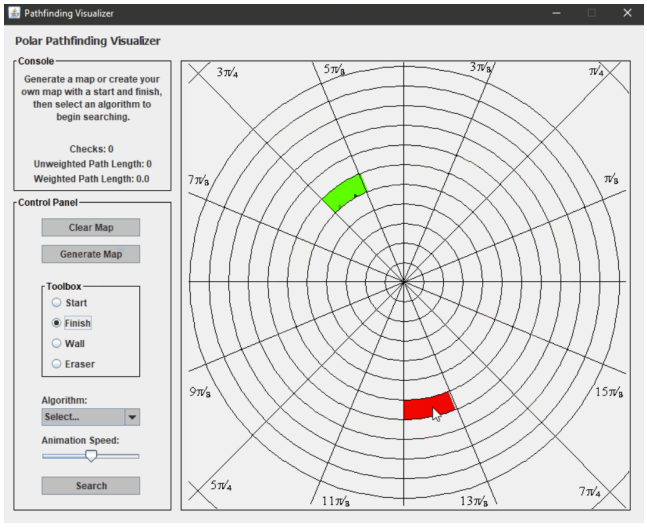

Figure 4: Console and Control Panel

4.4 Algorithms (BFS, DFS, and Dijkstra's)

The applet is capable of simulating three existing pathfinding algorithms: breadth-first search, depth-first search, and Dijkstra's algorithm. The following subsections review what they each do and then show images of each in action. The start and destination are green and red. Nodes that are checked are blue and nodes that are part of the path are yellow.

4.4.1 BFS

Breadth-first search explores equally in all directions, is unweighted, and guarantees the shortest unweighted path. Figure 5 shows BFS being simulated in the applet.

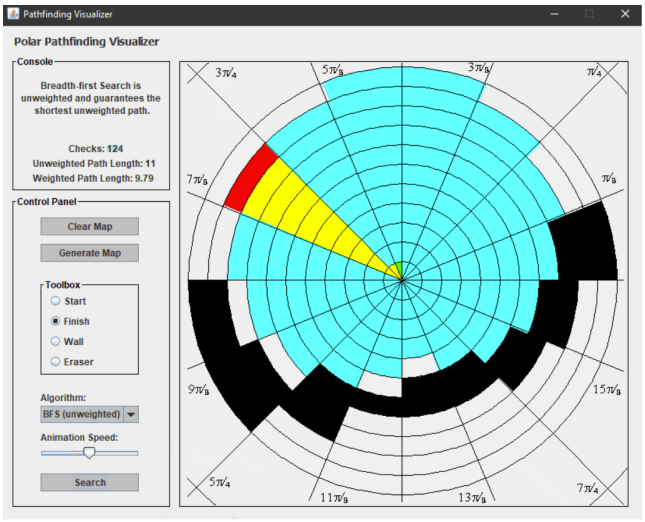

Figure 5: Simulation of Breadth-First Search

While journeying towards the destination, a fanning effect is observed.

4.4.2 DFS

Depth-first search traverses by exploring as far as possible down each path before backtracking. DFS is unweighted and does not guarantee the shortest unweighted or weighted path. Figure 6 shows DFS being simulated in the applet.

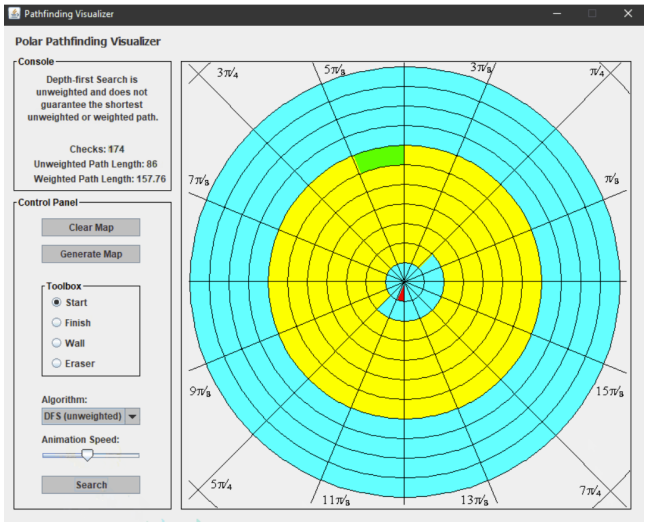

Figure 6: Simulation of Depth-First Search

4.4.3 Dijkstra's

Dijkstra's Algorithm prioritizes which paths to explore; instead of exploring all possible paths equally like BFS, it favors lower cost paths. Dijkstra's is weighted and guarantees the shortest weighted path. Figure 7 shows Dijkstra's being simulated in the applet.

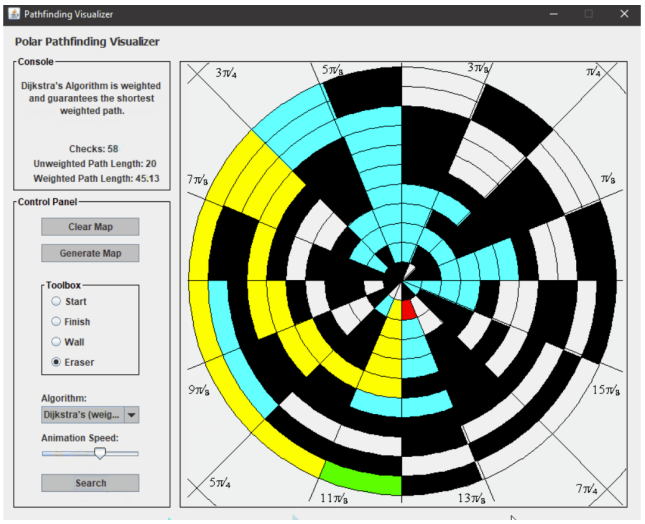

Figure 7: Simulation of Dijkstra's

5 Results

5.1 Comparing Algorithms

Performing step 2 of the procedure and making observations, this gif https://user-images.githubusercontent.com/72827220/

105569844-5793d500-5d13-11eb-9ee6-394efafc2ad3.gif shows each algorithm pathfinding for the same map.

As seen in the gif, BFS finds the shortest unweighted path of 11. However, it does not take weights into account and does not find the shortest weighted path (finds a weighted path of 20.96), traveling left before going towards the origin. Dijkstra's, on the other hand, takes weights into account and finds the shortest weighted path of 11.53 by traveling towards the origin before traveling left. DFS is very ineffective as its property to go down a path as far as possible before backtracking makes it go in a very long spiral, finding an unweighted path length of 133 and a weighted path length of 367.32.

5.2 Observations

Placing well-known route-planning algorithms in the simulator has led to some remarkable observations:

1. BFS does not take into account the arc length weights on the polar grid and can look misleading in the depiction. However, BFS is still able to find the shortest unweighted path represented by the number of edges
2. DFS' property to go as far down a path before backtracking makes it go in a spiral in a polar grid
3. As a weighted algorithm, watching Dijkstra's on the polar grid was able to find the shortest weighted path by journeying as close to the origin as possible before going left and right, recognizing how arc length is less with smaller radii closer to the origin

The visualizer makes it obvious why unweighted algorithms do not work on the weighted polar grid, leading to the discovery of an intuitive depiction of weighted versus unweighted algorithms using arc length.

6 Conclusion

This research has led to the creation of a Java applet useful for visualization, and numerous discoveries of graph algorithms in polar coordinates listed in section 5.2. Popular graph algorithms were placed in a unique environment, introducing a new perspective to their behavior.

7 Future Work

Although some progress has been made, going forward, more algorithms are to be implemented into the simulator, including A* and bidirectional search. Observations are still in the process of being converted into optimizations to make them more efficient and improve route planning in practice.

References

[1] Banerjee, N., Chakraborty, S. and Raman, V. Improved Space Efficient Algorithms for BFS, DFS and Applications. 2022.

[3] Dramski, M. Bi-directional search in route planning in navigation. Yadda.icm.edu.pl, 2022. https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-c9e1b9b9-ef5b-424b-b14d-58c264a5834f.

[2] Sanders, P. Algo2.iti.kit.edu, 2022. http://algo2.iti.kit.edu/schultes/hwy/schultes_diss.pdf.