Can The Spy Save The World? Understanding Route Optimization Using Dijkstra

Kushal Shah
University of Washington
Seattle, USA
kushals@uw.edu

Mohit Patel
University of Washington
Seattle, USA
mohitp@uw.edu

Jessica Hullman
University of Washington
Seattle, USA
jhullman@uw.edu

ABSTRACT

The use of animations to teach Computer Science Algorithms is a widely studied area of research [3]. Most algorithms require an advanced understanding of math and concepts such as graph theory to comprehend, hence increasing the knowledge barrier.

We develop an interactive visualization where users specify parameters and construct a randomly weighted grid. The users then specify origin and destination points on the grid, and the algorithm shows the shortest path the user can take to traverse from origin to destination on the grid. With this algorithm, we explore how users could be able to learn Dijkstra's algorithm with an interactive visualization.

INTRODUCTION

Route optimization algorithms to find the shortest and most cost affective path between two or more nodes in a network have variety of use-cases. Companies such as Uber are using these algorithms to pair users with nearby car-drivers to facilitate a ride-sharing ecosystem [4]

These algorithms are difficult to comprehend and require an understanding of graph-theory. This study aims to develop an interactive visualization to help a naive understand what Dijkstra's shortest path algorithm is. Dijkstra's algorithm is a technique for finding the shortest paths between nodes in a graph. This graph can be thought of as a road network where a car or a salesman has to reach its destination by incurring minimum cost. This algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956. There are multiple variations of this algorithm; Dijkstra's most common variant fixes a "source" node and computes shortest paths from the source to all other nodes in the graph, producing a shortest path tree.

Our aim was to develop an interactive visualization to make understanding of Dijkstra easier for the non-technical audience. Even the concepts that are used frequently are difficult to comprehend because of the inherent complexity associated with it. Learning difficult concepts have shown to result in higher attrition rates among students [6], hence leading to higher barriers to learning new concepts.

This acted as a motivation for us, to make it possible for the end user to vary input parameters and understand the mechanism behind Dijsktra. As examples aid in understanding algorithms [7], we created a story themed around a spy who has to deliver a secret package from one point to another and save the world.

The key sections of this study are:

1. Creation of a randomly weighted grid
2. Selecting Origin and Destination
3. Visualizing the fallacy of recursion
4. Constructing a graph of nodes and edges
5. Discovering the most cost-effective path

RELATED WORK

While surveying existing literature and attempts at making it easy to understanding the Dijkstra's algorithm, we came across multiple sources which attempt to explain how the algorithm works. From [2], we observed a detailed explanation about the theory behind the algorithm, covering concepts such as weighted graphs, directed and undirected graphs and greedy algorithms.

There was another attempt to help a person visually understand how the algorithm works [1]. The user can specify the start vertex, decide the graph type and chose the representation which they feel would help them understand the algorithm. The user has an option to decide the animation speed and the canvas size as well. The mechanism behind Dijkstra is displayed programmatically and is more suited to a technical audience.

Both the above sources are good for understanding the mechanism of the algorithm. We feet that by making the interactivity aspect slightly simpler by adding some context and easier transitions, the algorithm can be better explained to a non technical audience as well. This would help garner more appreciation and respect for solving hard complex computational problems by a wider audience.

METHODS

Dijkstra's Algorithm

Initially mark all nodes as a set of unvisited nodes. Each node is assigned a tentative distance value. That distance is set to zero for the initial node and infinity for all other nodes. The initial node is set as current node.

INFX 562

For the current node, consider all of its unvisited neighbors and calculate their tentative distances through it. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one as the new distance.

When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again. If the destination node is visited, the algorithm stops.

Algorithm 1 Dijkstra's algorithm

procedure DIJKSTRA($graph, source$)

  for each vertex in graph **do** ▷ Initialization
    $dist[v] \leftarrow Infinity$
    $previous[v] \leftarrow undefined$

  $dist[source] \leftarrow 0$
  $Q \leftarrow nodes$

  while $len(Q) \neq 0$ **do** ▷ Exploring all nodes
    $u \leftarrow$ node with smallest dist[]
    remove u from Q

    for all neighbors of u **do** ▷ Traversing neighbors
      $alt \leftarrow dist[u] + dist[u,v]$

      if alt < dist[v] **then** ▷ Updating distance
        $dist[v] \leftarrow alt$
        $previous[v] \leftarrow u$
  previous[]

Designing Interactive Visualization

The interactive visualization was developed using d3.js. The page was styled using HTML and CSS. We had to keep multiple things in mind while making the design decisions to lay out the web page for the users to understand the algorithm.

We divided the webpage into two sections. The left section is used more as a user guide for textual explanation to every corresponding step on the right section. The right section is used to display the visualizations which allow the user to specify their inputs and see the results.

In our initial design, we wanted to show the visuals side by side as that seemed more intuitive to the developers at the first glance. However, we had to divide the layout into text on the left half and visuals on the right half since the visuals were taking up a considerable amount of space.

The right section is divided into 3 main steps. Step 1 is where the user specifies the grid parameters to generate a grid on which the spy will traverse. We overlay the spy and the path traversal on the grid using a small blue circle with radius 7 for the spy and a black line for the path, both with 0.5 opacity. The opacity was chosen as 0.5 so that the weights are not hidden by the overlaid graphics.

We also made it possible for the user to click any box on the grid to specify the start and end points. The start point

7	7	8	4	6	1	0	3	6
2	3	6	5	5	1	2	5	2
0	5	6	5	8	2	3	6	3
5	8	6	5	4	0	6	5	3
5	4	3	2	3	8	4	3	6
9	1	4	0	6	3	5	2	7
2	6	7	5	8	3	9	5	3
8	9	6	9	1	0	7	7	2
7	8	9	1	8	2	8	8	4

Figure 1: Grid Construction

gets colored 'BlanchedAlmond' and the destination point gets colored 'lightgreen'. We also show the transition of the spy from origin to destination (or random destinations in case of brute force exploration) over a period of 3 seconds. This time frame was decided so as to allow the user to be able to follow the spy's journey across the grid but at the same time not be of an inappropriately long or short duration.

On clicking "Build Graph", a force directed graph with nodes and edges is generated with each node coloured black and sized according to their corresponding cost. The start node is colored crimson and destination node is colored darkgreen for easy identification for the user. Each node that is on the cheapest path transitions from black color to blue color when the user hits the "Find Cheapest Path" button. At the same time the grid in Figure 5 is generated that shows the spy reaching destination. These transitions take place simultaneously.

Narrative Storyboard

Keeping in mind a fairly non technical audience, we decided to develop an explorable explanation which would help the uninitiated to understand the problem with the context of a spy delivering a secret package to save the world.

Grid Construction The user specifies the number of rows and columns arbitrarily. Clicking the construct grid button would construct a grid of the dimensions specified by the user. The grid also displays in the form of text, the weight of each box in it i.e. the cost that would be incurred if our spy was to travel to that box from any grid.

Selecting Origin and Destination The user would then have an option to select two points as origin and destination in the grid. The first point that the user selects would be origin and it would be highlighted in red. The next point would be destination and that would be highlighted in green. The aim is to help the spy reach destination with minimum cost.

Exploring All Paths We would then ask users about the ways to find the optimal path between origin and destination. We begin by explaining the brute force method that the spy can

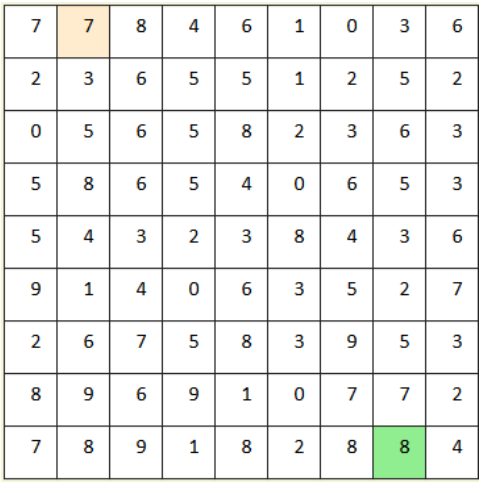

Figure 2: Selecting Origin and Destination

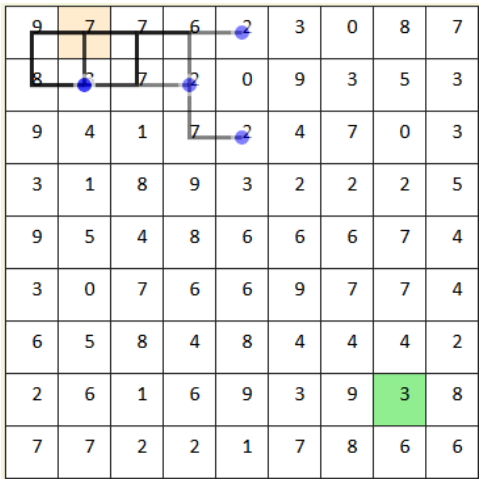

Figure 3: Exploring all paths

take, which is to calculate cost associated with all possible paths and then finding route with minimum cost. Users would then press the "Explore All Paths" button to visualize it.

This is a brute-force approach leading to multiple random paths of different costs and the spy will not be able to deliver the package in time.

Graph Transformation As part of the training, the spy is taught to think of this as a graph with nodes and edges. Each box in the grid is converted into a node and it is connected to its neighboring boxes (or nodes!) using edges. The size of each node is proportional to the weight or the cost associated with traversing through the node.

Shortest Path Now, the spy uses their friend Agent D's smart algorithm called Dijkstra's Algorithm to compute the cheapest path from source to their destination.

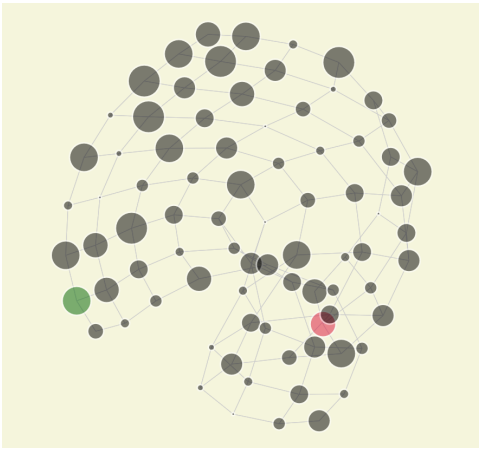

Figure 4: Graph Transformation

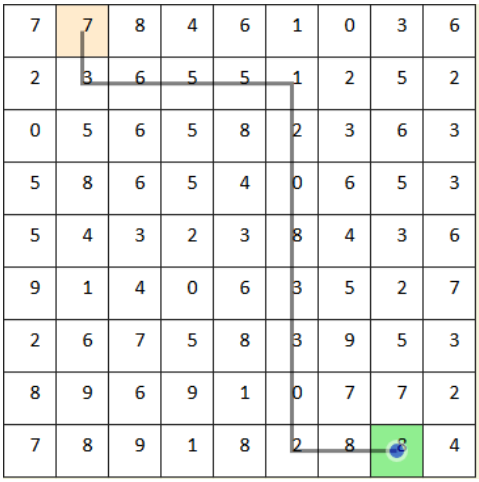

Figure 5: Shortest Path

RESULTS

We broke down the process to understand the algorithm into steps. The first step would be to start with specifying the number of rows and columns for the grid.

Figure 1 shows the grid visualization, displaying randomly generated weights within each square box as plain text. These weights are regenerated each time the grid is rendered on the canvas.

The user has the option of deciding the number of rows and columns and can also input the origin and destination by clicking a box on the grid.

Figure 2 shows that the origin point when clicked turns the box color to 'BlanchedAlmond' to indicate that it is the origin. The destination when clicked turns the box color to 'lightgreen', indicating that it is the destination point.

The user can explore all the possible random paths which can be taken on the grid. Figure 3 shows how the user can simulate a brute force exploration of the grid. When the user hits "Explore All Paths", a set of 5 spies and random paths appear, starting from the source and leading to different destinations to depict a confused spy desperately trying to reach their destination to deliver the package.

Then, we thought it would be interesting to show how this would look if visualized as a graph with nodes and edges where the cost to traverse to a node is denoted by its size. Bigger the node, higher the cost associated with traversing through it. Figure 4 shows the graph using the Force Directed Layout to simulate a creative approach to solving this problem. The force directed graph is a structure that shows the grid as a graph with nodes and links between neighbouring nodes.

The final step involves generating the cheapest path. We show this by regenerating the same grid as Figure 1 (see Figure 5), but, this time with only one spy and a single cheapest path leading from the source to the destination. When the user decides to explore the cheapest costing path from the start point to the end point, the nodes on the graph through which the spy has to traverse transitions from black color to blue color to demonstrate a path illumination phenomena. At the same time, as seen in Figure 5, the spy is shown navigating from the origin to the destination. As the spy reaches the end point, the total cost is displayed in textual format for the user.

DISCUSSION

Dijkstra's Algorithm using JavaScript

The constructed grid was a two dimensional array object with weights assigned to each index. Based on the origin and destination specified by the user, we had to derive the path having minimum cost to traverse from origin to destination along with the cost associated with that path.

According to the algorithm, we had to identify the unvisited node having minimum distance and then traverse all its neighbors and update the cost accordingly. So we created a dictionary object, where the object key was the cost to traverse from origin to that point and the value was an array of all nodes having that cost. At every iteration, we would fetch the node with minimum key and then push all it's unvisited neighbors into the queue for traversal. The node would then be deleted from the dictionary object. Another two-dimensional array was maintained to specify the current cost and the neighbor from which we can traverse.

Once the cost matrix was defined, we would loop over different nodes and find the path with minimum cost to go from origin to destination.

Algorithm Learnability

Our aim was to help users understand how a cost effective path could be computed using the two dimensional grid. The grid was useful to understand how different paths exist between two indices in a matrix. The complexity of traversing all combinations was explained when the user would click 'Explore All Paths' button. This would help the user understand that the problem is not as trivial as it looks and traversing all paths is not a cost effective solution.

The following graph would be useful for the user to understand how data-structures can be transformed from one state to another. Once the grid is transformed into a graph, users can immediately immediately understand that Dijsktra can help find the most cost effective path between two nodes. Given the dynamic nature of our visualization where users can specify the number of rows and columns and the starting and ending point, we decided to not show all the steps to compute Dijkstra.

The most cost effective path is then computed in the back-end and is visualized on both the graph and the grid.

FUTURE WORK

Visualizing Underlying Steps

The algorithm learnability could be improved if users can visualize how the shortest path is computed. We can visualize a cost matrix and show how weights are calculated with each iteration. Users can have an option to either watch this computation sequentially and can also skip all sequences to calculate the final cost. We can also show a queue data-structure to show the current node being processed.

Visualization Scalability

The visualization can be scaled to show different applications of the shortest path algorithm such as the "Traveling Salesman Problem" [5] Users can select multiple points on the matrix and the problems is to find the most cost effective route to visit all selected points on the matrix and to return to the original position. An ambitious add-on would be to design a ride-sharing system on a map, where users can request rides and the algorithm can try to match users with nearby drivers and other users requesting the ride.

REFERENCES

1. Dijkstra visualzation. `https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html`. (Accessed on 03/10/2018).
2. Finding the shortest path, with a little help from dijkstra. `https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e`.

3. Michael D Byrne, Richard Catrambone, and John T Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & education*, 33(4):253–278, 1999.
4. Judd Cramer and Alan B Krueger. Disruptive change in the taxi business: The case of uber. *American Economic Review*, 106(5):177–82, 2016.
5. Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.
6. Glenn Gordon Smith and David Ferguson. Student attrition in mathematics e-learning. *Australasian Journal of Educational Technology*, 21(3), 2005.
7. Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. In *International Conference on Machine Learning*, pages 421–429, 2016.