Visualizing Dijkstra's Algorithm Finding The Shortest Path

Final Project Report

Michael Uftring, Indiana University

Abstract. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph [wikipedia]. A path is a traversal along edges between connected nodes in the graph with the ultimate objective of connecting two endpoint nodes. Each edge traversal is also called a hop. Finding the shortest path generally serves the purpose of identifying the least costly traversal. Cost, in this sense, will emanate from one of two possible underlying conditions which are directly driven by whether the edges in the graph are weighted. In the case where the edges are not weighted, the cost would be derived from the number of hops between the endpoint nodes; the shortest path would have the fewest hops and thus the lowest cost. In the case where the edges are weighted, the cost of a path is the sum of the weights for each edge traversed; the shortest path would have the lowest sum total of weights (which may not be the same as the path with the fewest hops). This project will work to visualize networks, and the shortest path between two selected nodes.

Introduction

Background

Networks are everywhere. Visualizing networks is often challenging, especially if they are large (many nodes, and many links can clutter a display). Being able to visualize the shortest path between two nodes is often important for someone analyzing a network. The researcher may be attempting to determine the "cost" of routing computer network traffic, or determine the driving distance between two cities, or assess the impact of a "broken relationship" in a social network.

Motivation / Interest

The motivation for this project is two-fold:

First, I have a lot of interest in graph theory and networks. Through this course I have developed an even greater interest in data visualization. Combining these two interests seemed like a good idea, and a reasonable challenge.

Second, I wanted to undertake a project that was not a data analysis project. Instead, I wanted to take on a more technical (i.e., coding and implementation) project which would leverage what we have learned in class, and further my learning experience. Learning D3.js in this course was very valuable, and I am thoroughly impressed by its capabilities. This is another technical area that I wanted to exploit and learn more about.

To make the project a little more interesting and less trivial, I came to the idea of visualizing the shortest path in a network. This bit of inspiration came while looking at a subway map trying to find my way across New York City.

Objectives

The main objective of this project is to build a tool which will visualize networks, and visualize the shortest path between two user-selected nodes, and provide a means to interact with the visualization (i.e., not produce just a static visualization).

Some additional goals are:

- Provide a simple user experience
- Load the network from file
- Run Dijkstra's algorithm
- Show the shortest path by highlighting the source and destination nodes, and the traversed edges
- Produce a summary table

Some "stretch" goals were:

- animate the visualization as the algorithm executed
- Allow interactive creation and editing of a network
- Save the network to a file

The amount of time available and my limited knowledge in some areas (web applications, and JavaScript) proved the stretch goals a little too much to take on and complete this semester. However, I still keep this set of goals and objectives and see the possibility of working further on this project in the future (even if just for my personal benefit).

Related Work

There are a lot of existing network visualization tools and applications on the Internet, and many of them provide some form of visualizing Dijkstra's (or other shortest path) algorithm. What follows is a short summary of five, where I present my "Likes" and "Dislikes."

David Galles, University of San Francisco

https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Likes:

- The animation by this application is very clear and straightforward. It highlights the nodes and edges as the algorithm executes, and is very nicely done.
- The ability to pause, and step algorithm execution (though it is slightly confusing) is very useful when trying to follow along and understand what's happening.
- The tabular results when execution is complete. Is shows the cost from the starting node to each reachable node, and it shows the hops (edge traversals).

Dislikes:

- Graph origin is only through random generation by the application itself. There is no means to provide a user-defined graph, and no means to edit a generated graph.
- It does not allow you to select start and end nodes.
- Thus, it does not visualize (in its logical representation) the shortest path between two selected nodes

VisuAlgo.net

https://visualgo.net/en/sssp

Likes:

- The lecture material on various algorithms makes this a very useful learning tool, as well as a means to visualize Dijkstra's and 4 other algorithms.
- The animation: highlighting on nodes and edges as the algorithm executes is really neat. The way the edges are highlighted shows clearly the direction of each edge.
- This application also allows stepping through algorithm execution. It

Dislikes:

- Interactive graph editor is clunky, particularly when you want to remove an element. Also, the editor does not permit moving nodes.
- The mechanism for selecting the starting node is strange, and at first not very apparent. When you select an algorithm from the menu it shows a pop-out with an `s = [number]` selector.
- It does not clearly show the results. The path distance from the starting node to all other reachable nodes is displayed just below each node with red text. When placed on top of an orange colored edge it is very difficult to see..

GraphOnline.ru

http://graphonline.ru/en/

Likes:

- Interactive graph editor allows moving nodes around (wherever you want)
- It also has an “Arrange the graph” option, which arranges the nodes and edges in a neat manner
- Zoom in and out capability
- Can change node size based on weight

Dislikes:

- Creating edges in graph editor felt clumsy. When you connect two nodes it presents a dialog box from which you specify the edge weight.
-

Lisa Velden, Technischen Universität München

https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_en.html

Likes:

- This application is a complete learning experience. It provides a very good explanation of Dijkstra’s algorithm with some visuals.
- The graph editing capability was simple and straightforward. You can edit an existing graph, or create a new one.
- The ability to Upload and Download graph definitions.
- This application also provides the ability to step through the algorithm, both visually and descriptively.

Dislikes:

- A minor critique is the graph representation format. It is a simple text-based node and edge definition. While likely efficient from a space perspective, it is not easily human readable. However, given the nice editing capabilities this could be easily overlooked.

Xueqiao (Joe) Xu, Google

https://qiao.github.io/PathFinding.js/visual/

The focus of this project is on finding paths in grid-based games. At first, it appears that it does not operate on a network. But after using the tool a few times and reading some supporting material in the GitHub repo, I believe that the entire grid is a connected network and the obstacles and options essentially remove nodes and edges.

Likes:

- It provides seven different algorithms to select from, and permits setting of some parameters.
- The animation, given the 2-D gaming perspective, is kind of cool; this is especially true of the “search-based” algorithms.
- This web application presented a whole new dimension to networks and visualization. It was a welcome “distraction” while researching related material.

Dislikes:

- None really, but this one is not really applicable to this project and its objectives.

Research / Questions

There were no real questions, per se, being addressed by this project. At least not those which would be more typical of a data analysis and visualization project. The questions were mostly of a technical nature:

(1) How best to represent the networks?
    (a) Here, I opted to use the Network JSON format that was used in the course lab exercises. This format seemed comprehensive, and easily extensible (if needed, which it has not so far). This format also has the benefit of working easily with D3.js we learned this in one of the labs.
(2)

The main avenues of "research" for this project were along three lines:

1. Dijkstra's algorithm
2. JavaScript
3. D3.js

I learned about Dijkstra's algorithm and finding the shortest path in a graph during undergrad study (many years ago). This I needed to refresh my memory, and found several good resources online aside from the web applications which are visualizing networks and demonstrating execution. The Wikipedia article has a good overview and discussion, along with some of the underlying mathematics.

Gilles Bertrand has written two blog articles on Dijkstra's algorithm. The first one [bertrand-1] provides a good description and a step-by-step example with hand drawn images. The second article [bertrand-2] provides a Python implementation which is concise and very easy to follow. This code was adopted and adapted for this project.

The blog article written by Vaidehi Joshi [joshi] has a very modern feel to it, and is also well written with nice visuals. She has written a series of articles across several topic areas which are well worth looking into for learning and relearning (as was my need).

Process

This project was divided into three stages:

(1) Research: finding descriptive and background material on Dijkstra's algorithm, including existing visualization tools and applications.
(2) Exploration: the process of relearning Dijkstra's algorithm through hands-on experience.
(3) Implementation: building the final work, a web-based application which will visualize networks and the shortest path between two nodes

The research portion is

Exploration

The exploration phase was performed with Jupyter Notebook. The objective was to relearn Dijkstra's algorithm, and step through several example networks to solidify my understanding. I imported and adapted Gilles Bertrand's Python implementation. I added "debug" statements so it would produce a running dialog of execution which could be read and understood.

Next was to attempt to visualize the network in the Jupyter environment. This proved to be more challenging than expected. NetworkX and Matplotlib are very good network and visualization packages respectively, and they can be made to work well together. However, the visualizations produced are static and do not provide any interactive nature.

I found the visJS2jupyter package which promised to be: "a tool to bring the interactivity of networks created with vis.js into jupyter notebook cells." The introductory blog post [visJS2jupyter] was enlightening and really helpful. The example provided worked "as-is" and without any hitches, once the package was installed. It was also easy to adapt for the example network I was using. This gave a nice visualization of the network, and some interactivity. If I had more time during this project, I would had attempted to visualize the shortest path using this tool in Jupyter.

Implementation

The implementation was done in an incremental fashion, adding small amounts of capability at a time and ensuring it worked as desired before moving on to the next.

The basic development plan was:

- Visualize a network with D3.js (this was work from Lab 15)
- Load the network from a file
- Figure out how to get user selections for start and end nodes
- Figure out how to "save" those selections so they could be used by Dijkstra's algorithm
- Implement Dijkstra's algorithm in JavaScript
- Figure out how to highlight the results
    - Start node = green
    - End node = red

    - Intermediate nodes = yellow
    - Traversed edges = red
- Figure out how to produce tabulated summary of results
- Figure out how to allow user to “start over”

Implementing most of these capabilities entailed some amount of online search and learning. Where a blog or Stack Overflow article contributed significantly, it is cited in the code. Porting Gilles Bertrand’s Python implementation of Dijkstra’s algorithm to JavaScript was actually a lot easier than expected. The syntactical similarities of the two languages certainly helped.

Results / Insights

See:

- Jupyter notebooks
- Web application (HTML file)

Discussion / Conclusion / Future Work

I learned a lot working on this project, and had a lot of fun doing so. My refreshed knowledge of Dijkstra’s algorithm will certainly be helpful in future networking and graph projects. I have also learned quite a bit about D3.js, though it appears there is still quite a lot to learn. One of the best print-based resources I found was a very recently published book: D3.js in Action, Second Edition [meeks2ed]. This book, while not yet completely read, takes a really useful and practical approach to teaching D3.js. It is not just “how to” but also provides answers to many “why would you..” type of questions. For an experienced software developer new to D3.js, this is a great reference.

What’s next? There are a lot of things which I wanted to do with this application. But given this is the first non-trivial web application that I have written (i.e., beyond a “hello, world”), I am mostly satisfied. However, it is clear that I’m not yet an experienced web developer or D3.js expert. With my interest in graphs and networks, and desire to learn more in this space, and newfound growing interest in data visualization, I expect to keep working on this application -- even if just for personal enrichment.

Future work:

- Animate the algorithm execution
- Display the node names
- Display the edge weights
- Show directed edges (with arrows)
- Allow zoom in and out
- Provide search for finding start and end nodes
- Allow interactive editing of the graph

- Save results
- Explore visualizing Shortest Path in Jupyter Notebook with visJS2jupyter
  (consider implementing some capability and submitting a Pull Request to the GitHub project)

References

[bertrand-1]	http://www.gilles-bertrand.com/2014/03/disjkstra-algorithm-description-shortest-path-pseudo-code-data-structure-example-image.html
[bertrand-2]	http://www.gilles-bertrand.com/2014/03/dijkstra-algorithm-python-example-source-code-shortest-path.html
[joshi]	https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e
[meeks2ed]	https://www.manning.com/books/d3js-in-action-second-edition
[visJS2jupyter]	http://compbio.ucsd.edu/bringing-interactivity-network-visualization-jupyter-notebooks-visjs2jupyter/
[wikipedia]	https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Additional Material

(not cited, but worth reading for additional background)

[brilliant]	https://brilliant.org/wiki/dijkstras-short-path-finder/
[geeks4geeks]	http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
[krishtopa]	https://steemit.com/popularscience/@krishtopa/dijkstra-s-algorithm-of-finding-optimal-paths
[rosetta-code]	https://rosettacode.org/wiki/Dijkstra%27s_algorithm
[sci-direct]	http://www.sciencedirect.com/science/article/pii/0020019077900023?via%3Dihub