SHORTEST PATH ANALYSIS

A PROJECT REPORT

December 1, 2025

TEAM MEMBERS

David Slate Lee

Connor Buttrey

AJ Wood

Kai Matton

CSC 2400: Design of Algorithms

Final Project

Overview Of Problems And Methods

The main goal of this project is to compute, test, and compare shortest path algorithms on large-scale, real-world road network graphs. Our dataset is a weighted, directed graph that represents the New York City road network, containing 264,346 nodes and 733,846 arcs. Determining the shortest path between pairs of nodes is useful for navigation systems (e.g., Google Maps, Apple Maps), ride-sharing and delivery companies (e.g., Uber, Lyft), and network analysis (e.g., ESRI, SafeGraph). Because real road graphs are often sparse, positively weighted, and geographically structured, they present both opportunities and challenges for algorithmic analysis, making this an ideal area for research.

To study this problem thoroughly, we intend to implement and evaluate four well-known shortest-path algorithms: Dijkstra's Algorithm, A*, Directed Acyclic Graph (DAG), and Bellman-Ford. Each method represents a different design approach and therefore serves to help achieve a well-rounded analysis.

Objectives & Research Questions

Our goal is to test, compute, and compare shortest path algorithms on large-scale, real-world road network graphs.

- What are the time comparisons between nodes 1-200 and 1-100,000?
- What are the differing path lengths between nodes 1-200 and 1-100,000?
- What are the differing path costs between nodes 1-200 and 1-100,000?

We expect that the Bellman-Ford will be the slowest due to the time complexity, especially for nodes from 1-100,000. The A* algorithm is expected to be the quickest overall due to its use of heuristics.

Description Of Experiments

We are choosing to use a Jupyter Notebook file to host all of our code through Google Colab and will be coding in Python. This can be accessed here: We are testing the shortest path algorithm on the dataset that contains real nodes of New York City and its roads, which contains 264,346 nodes and 733,846 directed edges, with the weight being represented by the distance in meters. The graph is stored as an adjacency list. We will use four separate types of shortest path algorithms (A*, Bellman-Ford, Dijkstra) on the dataset and analyse their efficiency with a dataset of this scale. With these algorithms, they will be run on a short distance pair (1-200 nodes) and a long distance pair (1-100,000). Additionally, with the help of generative AI (specifically ChatGPT), the coding process and debugging processes will be much smoother. The computer used contained a 13th Gen Intel(R) Core(TM) i7-1360P 2.20 GHz processor, 16.0 GB (15.6 GB usable) of installed RAM, and a 64-bit operating system for its system type.

Results

We originally planned to include the Directed Acyclic Graph (DAG) shortest-path algorithm in our performance comparison, but it could not fit our dataset. Real-world road networks contain many cycles due to intersections and loops. Because a topological sort is impossible on a cyclic graph, we were not able to compare DAG to the rest of the algorithms.

However, since Dijkstra, A*, and Bellman-Ford all interacted well with our dataset, we were able to collect a lot of information on the path and runtime of these algorithms. The table below contains the information we were able to collect after averaging our results on a single team member's laptop. First, our results on testing a shortest path from node 1 to 200, and second, our results on testing a shortest path from node 1 to 100,000:

Nodes 1 → 200			
	Path length	Total Path Cost	Total Time Taken
A* Algorithm	47	88037 meters	0.001470 seconds
Dijkstra's algorithm	47	88037 meters	0.074190 seconds
Bellman-Ford Algorithm	47	88037 meters	1.609913 seconds

Nodes 1 → 100,000			
	Path length	Total Path Cost	Total Time Taken
A* Algorithm	874	960819 meters	0.198317 seconds
Dijkstra's algorithm	814	955712 meters	0.8495 seconds
Bellman-Ford Algorithm	814	955712 meters	17.226191seconds

Interpretation

Nodes 1 -> 200. All three algorithms had the same path length. This shows that all algorithms were able to find the shortest path. A* was by far the fastest: 0.00147 seconds. Dijkstra took 0.074 seconds, around 50× slower than A* in this test. Bellman-Ford was the slowest: 1.61 seconds, roughly 1000× slower than A*. On small graphs, even though all algorithms found the same path, A* is by far the most efficient of the three. The massive jump in time taken with Bellman-Ford shows how computationally heavy it is, even at a low count of nodes.

Nodes 1-> 100,000: All algorithms find valid paths, though the lengths and costs tend to differ when the dataset grows. Multiple near-optimal options exist within the graph. A* was still the fastest at 0.1983, even though it cost more. Dijkstra is still taking second place in speed, but it was more efficient in its cost and path length. Bellman-Ford remains in last by still costing more and taking roughly 50x longer than Dijkstra. However, it crossed fewer nodes. But due to it costing more, it is still not more efficient than A* or Dijkstra. It should also be noted that Dijkstra had the cheapest path cost, therefore meaning it contained the shortest path. Additionally, even though A* should've been the fastest in its total path cost, it wasn't, which could be credited to how it was implemented.

Key Takeaways: Our results clearly fit the known time complexities of the algorithms, since both Dijkstra and A* have a significantly shorter runtime than Bellman-Ford. Dijkstra and A* have a time complexity of O((V+E) log V), with A* also guided by a heuristic. And Bellman-Ford has a time complexity of O(VE). A* is the clear winner for both the smaller and larger datasets that were used. Dijkstra is reliable and reasonably fast, but it becomes slower as the graph size grows. Bellman-Ford is too slow for larger-scale needs. However, it is still your best option if negative weights exist due to A* and Dijkstra's inability to work with those.

Sources

- Dataset (distance graph & coordinates)
    - https://www.diag.uniroma1.it/challenge9/download.shtml

- SHP (Shapefile) file. This was used to create some of the visuals for the slides where the nodes are shown on a real outline of Pennsylvania, New York, and Connecticut:
    - https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-1-states-provinces/
- GeeksForGeeks. We used the following articles to get an understanding of the concepts within the project:
    - https://www.geeksforgeeks.org/dsa/shortest-path-for-directed-acyclic-graphs/
    - https://www.geeksforgeeks.org/dsa/a-search-algorithm/
    - https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/
- ChatGPT
    - AJ: I used ChatGPT to help explain how A* works, how to implement it with such a large data set, and how to make it visualise
    - Slate: I used ChatGPT as assistance in implementing Dijkstra's algorithm.
    - Kai: I used ChatGPT as assistance in implementing the Bellman-Ford algorithm.

Description Of Work

Task	Name	Percent
Brainstorming	AJ, Connor, Slate, Kai	25% each
Coding	AJ, Connor, Slate, Kai	25% each
Experiment Design	AJ, Connor, Slate, Kai	25% each
Data Collecting	Connor, Slate	50% each
Visualization	AJ, Connor, Slate, Kai	25% each
Report Writing	AJ, Connor, Slate, Kai	25% each
Organization	Slate	100%
Communication	AJ, Connor, Slate, Kai	25% each
Presentation	AJ, Connor, Slate, Kai	25% each
YouTube Video Editing	Slate	100%

Difficulties & Roadblocks

In this project, we faced many roadblocks and difficulties that took teamwork to overcome. One of the first challenges that we faced was finding an appropriate dataset to effectively test and compare our shortest path algorithms. Because of the diverse characteristics of each algorithm, we needed to look for a dataset that is large enough and structured in a way that we can display the pros and cons of each algorithm.

In the beginning, we figured that we could use a small dataset in which we just use a maze to represent the nodes, edges, and vertices. We later found that this was going to be too small or simple, which would make it difficult to make an observation that shows accurate runtime, efficiency, and path-finding quality. As a result, this roadblock meant a large portion of our time was spent looking for an efficient dataset, in which we eventually landed on a dataset that represented a city in New York. In this dataset, we found essential components such as edges, nodes, distance, and vertices.

Further on into the project, we were able to parse the dataset into our program, but the main issue we had next was the implementation of the DAG algorithm. Due to the nature of DAG being acyclic, this caused an issue because our data set had cycles. Our team talked and figured that we should create a synthesised dataset that resembled our New York dataset, so we can make an analysis of the algorithm. After the analysis of the synthesized data we discovered that the difference between the datasets was too great to make an impactful comparison to other algorithms.

In conclusion, overcoming this roadblock helped us understand how important dataset selection is when we analyse algorithms. Choosing the right data set was important to reveal the pros and cons of each shortest path algorithm. As a group, one major real-world takeaway is that we can think of this issue similarly to scalability. In this project, we wouldn't have noticed the difference between algorithms if we chose a smaller dataset, similar to the real world. If we built an application for a smaller number of users, it may seem efficient at first, but as we scale, the performance issues could appear a lot quicker.