EC504 Final Project Report
Route Recommendation Application

Jung In Chang
Taehyon Paik
Minseok Sakong
Yuxuan Luo

Github link: http://github.com/changju784/Route_Recommendation_App

Team Information

	Task	BU ID	SCC username
Jung In Chang	Dijkstra's Algorithm, Dataset, Project Organization	U07196971	changju
Minseok Sakong	Building Graph, Constructing Node Class/Property	U52516047	msakong
Taehyon Paik	A* Algorithm (Diagonal Distance, Great Circle Distance), Driver Function	U58182574	ttpaik
Yuxuan Luo	Front-end and server (GUI)	U70644612	luoyx

Abstract

This project will be focused on building an algorithm for a real-time navigation system. Using given coordinates of map in the form of network data with longitude and latitude coordinates, our goal is to test different algorithms including Dijkstra's and A* search to extract the shortest path from a source to destination. Final deliverable will be a clear GUI of web application containing these features.

The python-based program generates a graph of Boston map using OSMnx package to download geospatial data from OpenStreetMap API. It converts streets and building networks to vertices and edges to form the map in a graph format. It receives coordinates of source and destination from the user and sets them as starting and ending nodes. By calculating length of edges using different implemented algorithms, it plots shortest paths on the map with different colors. The result of the plotted graph is stored as output and visualized in the web application.

Instructions

Graph Construction

In order to build a graph containing building and street information, the OSMnx Python package was used. In build_graph.py, OSMnx conducts an OpenStreetMap API call and retrieves a city map with street network information. Downloaded as a graphml file and stored under the “dataset” directory, the program can load the Boston map and run the shortest path algorithm.

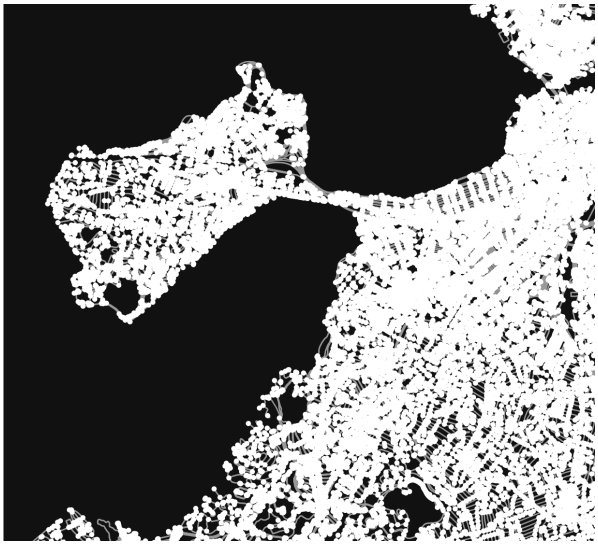


Fig 1. Boston Map using OpenStreet API

First, we convert the latitude and longitude of source/destination locations to the nearest node in the graphml. The euclidean distance algorithm is used to find the nearest neighbor nodes. After retrieving the source and destination nodes in the map, the program checks whether there is a path between the two points using a predefined path calculation algorithm in the OSMnx package. This prevents an infinite loop from happening in the path-finding algorithms.

After the sanity check, the graphml file is converted into a directed graph with vertices and weighted edges. From the graph, latitude and longitude of source and destination location

is translated into the nearest node. The Euclidean distance algorithm is used to find the nearest neighboring nodes. With data stored as iterable variable formats, our program finds the shortest path and the shortest distance between the two points.

Algorithms

The shortest path from the starting node to the end node is implemented and tested with different algorithms using the previously fetched constructed graph.

- Dijkstra's algorithm

  Under engines/algorithms/dijkstra.py, the algorithm iterates the graph adjacency object holding the successors of each node. A min heap is used to store distances from each node to its neighbors. By popping the root node from the heap, it was able to extract the shortest distances. In every iteration, visited nodes are marked using hashmap to avoid revisiting. Another hashmap of distances is set, which contains nodes and distance in a key-value format to accumulate the shortest distances. If the node is not visited and contains a shorter distance than the current cost, the distance hashmap will be updated and heap elements will be pushed. Lastly, nodes are stored in a list backwards starting from the target.

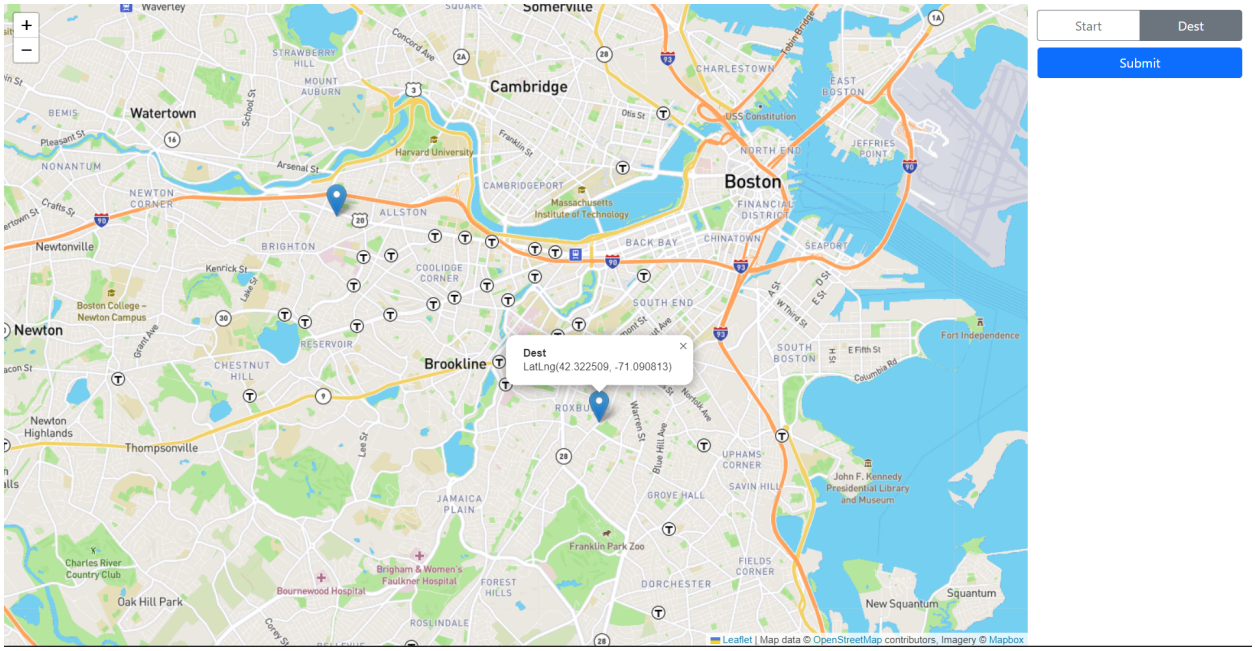
- A* Search

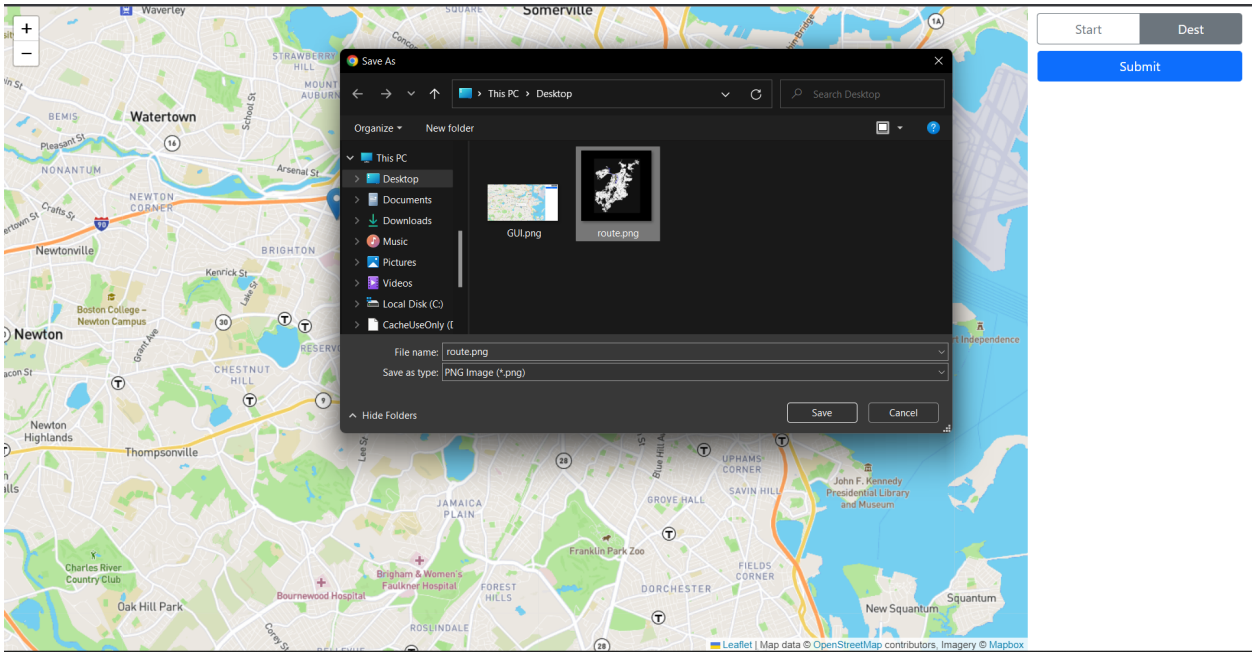
  Under engines/algorithms/Astar.py, The A* algorithm is written based on the dijkstra's algorithm, but a more informed version. A* uses BFS and chooses vertexes to explore based on the Heuristic Function, and the Cost. Generally, the Euclidean and Manhattan Heuristic functions are used to compose the algorithm. However, for the purpose of navigating through roads that are more than 4 directions, our version uses the Diagonal Distance Heuristic Function, and the Great Circle Distance Heuristic Function. Here, the Diagonal Distance Function adds accuracy when there

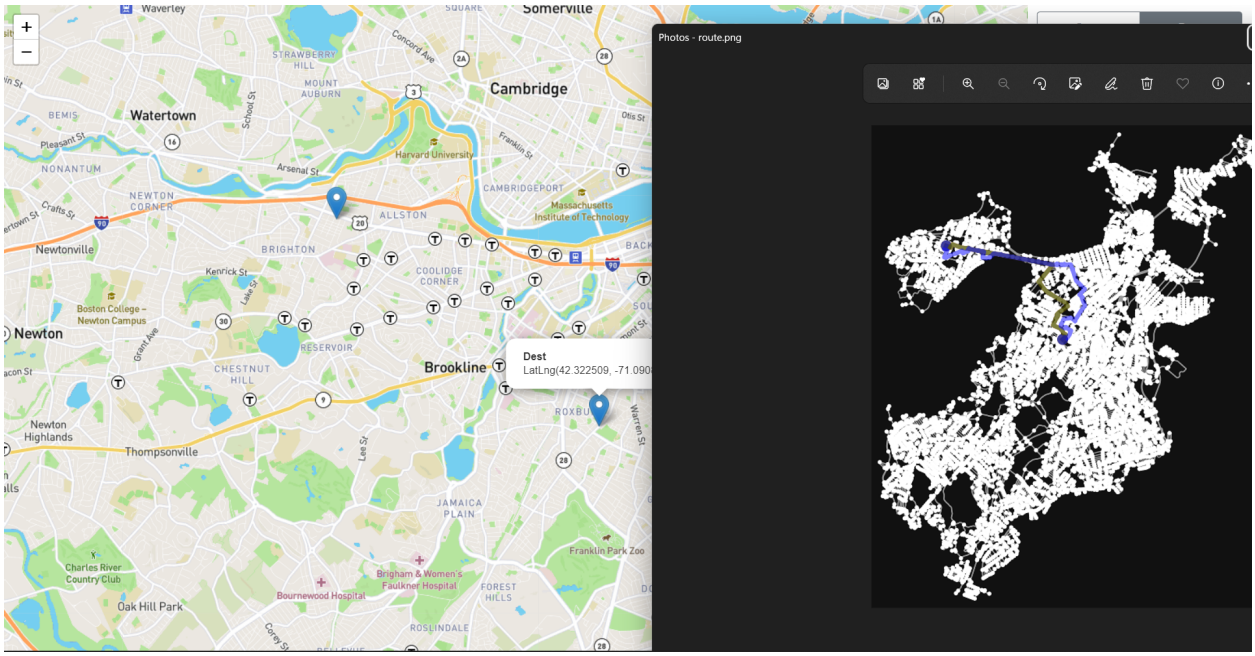
are 8 available directions to move, whereas the Great Circle Distance covers the fact that a distance between two points on Earth's surface will not be a straight line.

Web Application

The web application is the GUI to this algorithm. It consists of two parts: frontend and backend. The server is written in Python with the support of Flask framework so it coordinates with the algorithm codes smoothly. The client is a HTML file using Leaflet and Bootstrap libraries; it prompts the users to pin the start location and the destination, then it sends these coordinates to the server. After the server handles the data and renders the image, the client receives it and stores it on the disk.




Start
Dest
Submit
Save As
This PC › Desktop
Search Desktop
Organize
New folder
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
CacheUseOnly (I
GUI.png
route.png
File name: route.png
Save as type: PNG Image (*.png)
Hide Folders
Save
Cancel
Leaflet | Map data © OpenStreetMap contributors, Imagery © Mapbox


Photos - route.png
Dest
LatLng(42.322509, -71.090

Sample Results

By running main.py, user can obtain plotted graph showing three different algorithms (Dijkstra, A* search diagonal, A* search great circle) explained above. Source and target coordinates can be adjusted by editing input/input.txt so that the program reads source and target longitude/latitude. Dijkstra is colored by red and each A* search in blue and green.

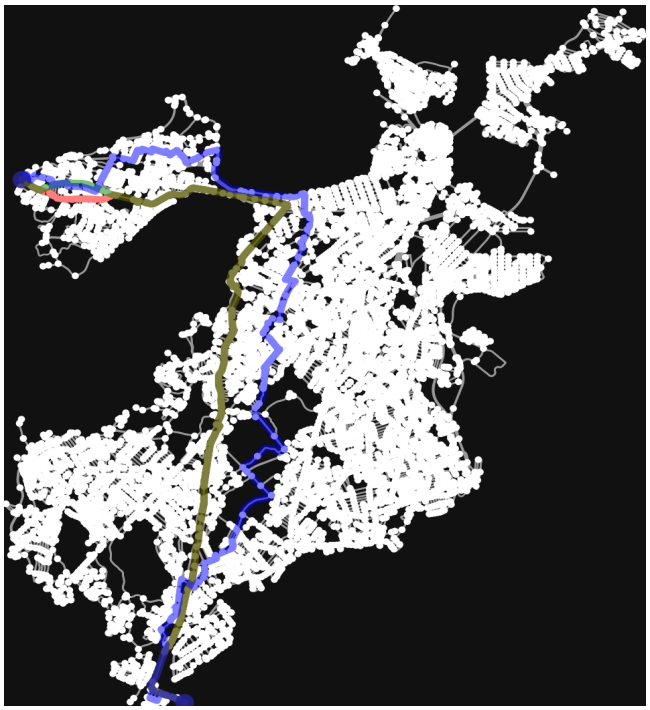


Fig 2. Shortest path with different algorithms (using "Boston.graphml")

This however, is not the optimal shortest path from source to destination. This image uses the "Boston.graphml" file, which is a smaller version of our original file "bigBoston.graphml" file. As it can be seen, there are roads that exist, but are not shown in the diminished map. When using the actual map, results are shown as below.

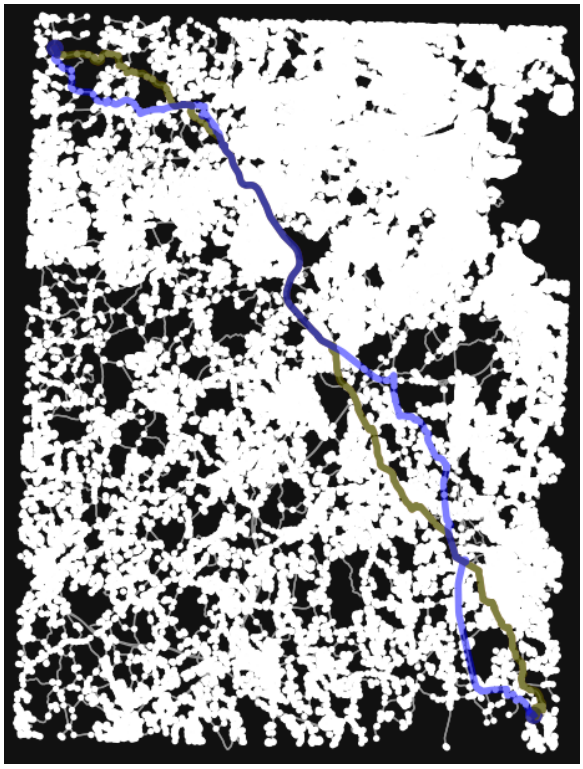


Fig 3. Shortest path with different algorithms using ("bigBoston.graphml")

The map shown above is a test case that uses coordinates: Source (42.0, -71.0) Destination (42.4, -71.4). Comparing this with Google maps, the image below shows Google's shortest path using the same coordinates.

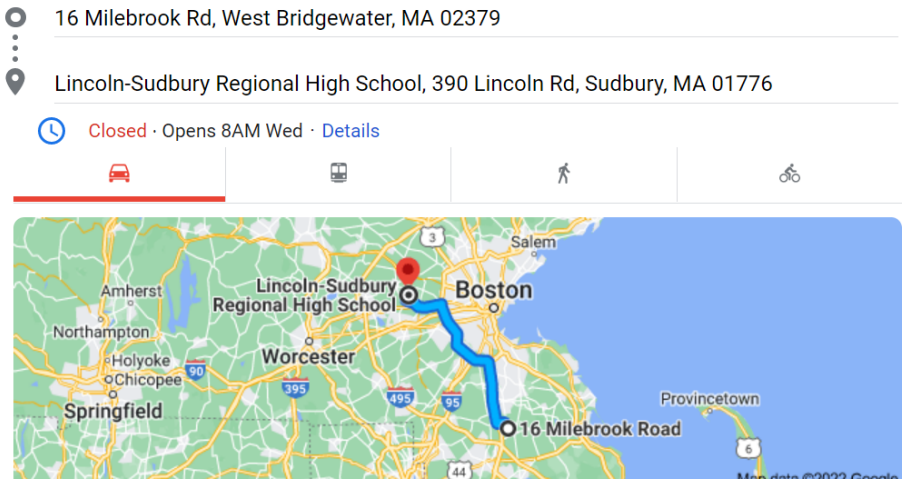


Fig 4. Shortest path from Google Maps

The results of our shortest path algorithm are very similar to what the actual Google maps shortest path indicates. The distance calculated from google maps was 43.6 miles, which also similarly aligns with our algorithm's results as shown below.

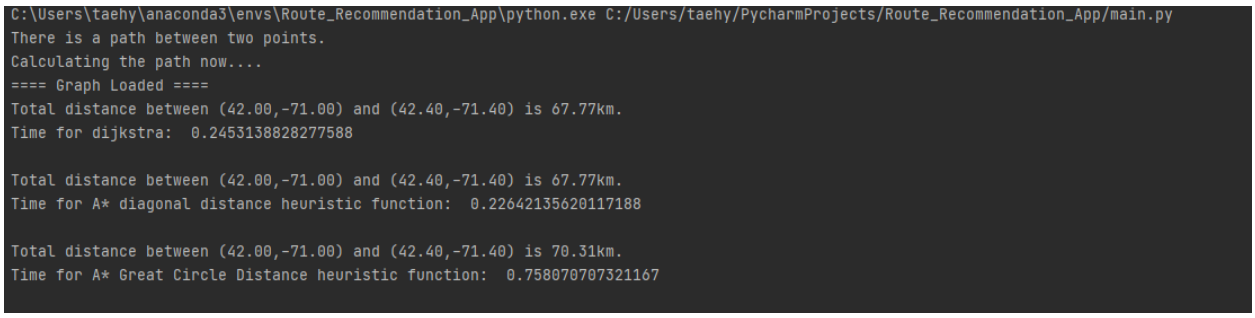


Fig 5. Results from shortest path algorithms using ("bigBoston.graphml")

The results show an average of 70km for the shortest path from source to destination. Google's estimated distance 43.6 miles is equivalent to 70.1674 km, which shows high accuracy of our algorithms.

References

- OSMnx Python for Street Networks: https://geoffboeing.com/2016/11/osmnx-python-street-networks/
- Dijkstra's algorithm: https://www.programiz.com/dsa/dijkstra-algorithm
- A* Algorithm: https://www.geeksforgeeks.org/a-search-algorithm/
- Great Circle Distance Formula https://www.geeksforgeeks.org/great-circle-distance-formula/
- Leaflet https://leafletjs.com/
- Bootstrap https://getbootstrap.com/