EleNa: Elevation-based Navigation
Final Report

POIUYT Team

1. **Problem description:**
    - When using navigation to traverse from one source to another destination, it is common to use the shortest or fastest path. For some customers like sport practitioner, runner, racer, they have additional requirements for workout purposes, however, the elevation will be considered except from altitude and longitude. If customers want to maximize the effectiveness of consuming more calories or selecting special types of workout intention of climbing up and down, the elevation would be used. In this project, we develop a software system for navigating from a source to a destination considering maximum and minimum elevation with a percentage x of shortest path length.

2. **Model design:**

   The high–level architecture is shown in Figure 1:

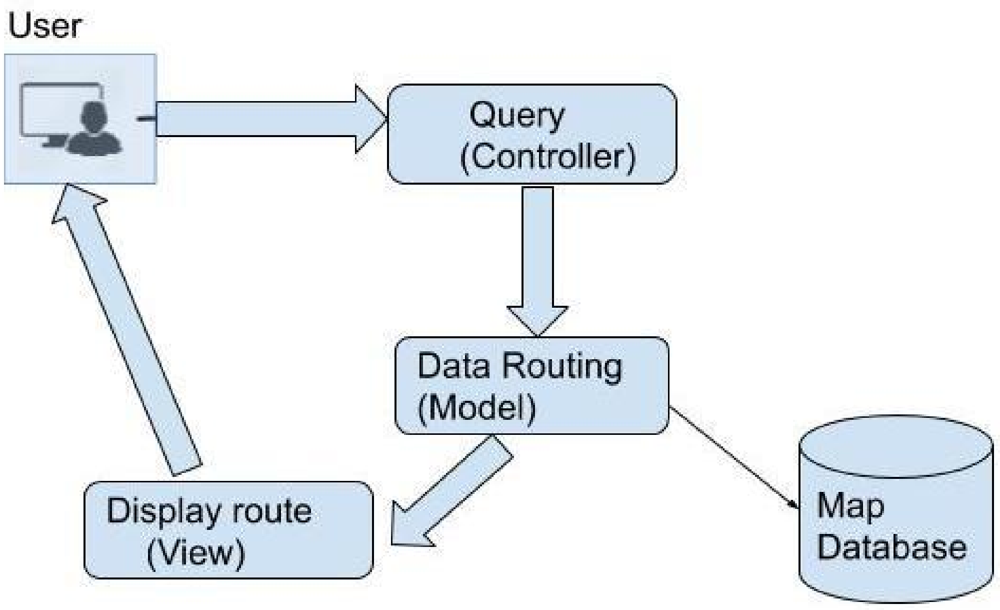


Figure 1: Architecture of EleNa navigation system

- navigation system as Desktop front end user interface and backend.
- Data model: Built in graph model in OSMNX
- Front end: We display an interface for query to populate the data model with an input of;

i. source and destination of address.
ii. The shortest path percentage
iii. The max or min elevation button selection.
iv. The map (offline map) and the output route shown on the interface
v. Pop-up Web Browser showing routing

- Back end: We store the geo data locally from OpenStreetMap about longitude and altitude and elevation in the pickle file. Then we insert elevation data from Google Elevation API to pickle files for further model generation.
- **Proposed Algorithms:**

(1) one proposed way is to use a modified Dijkstra algorithm.

i. For example if we consider min elevation in a x percentage of shortest path.
ii. In the iteration of the Dijkstra algorithm, in each iteration, we select a node that has the shortest distance to source as a starting node. Here we select a node with smallest elevation to iterate, and then calculate the shortest paths around the node neighbor, in the next iteration, we continue to select the node with smallest elevation and calculate the shortest paths around the node neighbors, and so on.

(2) Second algorithm is A* search: we will use shortest distance and straight-line distance as our heuristic in the design. Given a source and target node, each node has f, g, h values, where f is the total distance to the goal, g is the heuristic distance to the goal, h is the cost from the source. The edge weight between node v and node u has two values, the elevation difference and distance w(u,v) = [elev, dist]. For example, given a min elevation with source minimum elevation and a shortest distance percentage x, During the running of A*, each time we pick the node with minimum elevation and then shortest distance to expand and traverse . We keep the travsering until we find a path that is close to shortest distance percentage x, we stop the travsering and retrieve back the path, which is the route we need.

(3) BFS: we tried to develop BFS and DFS for this graph searching but there are a few constraints that make this algorithm nesty.

- We can not mark unvisited and visited nodes since one node can be visited by different routes. We have to make one route that will not visit a node twice to avoid cycles.
- It is still super slow if the allowed cost is high and we search 10 miles far away even if we drop solutions as it beyonds allowed cost.
- We have to record every route if they are possible to reach the target for tracking.
- There is no way to gather all possible routes and return a best one. It is far beyond our time scope.

3. Architecture of our frontline system:

The system interface architecture is show in the following Figure :

4. **Use cases:**
   As referenced in the demo video, the user can input the origin and destination address in the Amherst, MA (currently the database is tested in limited in Amherst only as the original database only contains Amherst, we can use different cities to download in our backend if needed).
   Except from this, there are maximum, minimum elevation and shortest distance percentage options. Users can use the options to decide a route generated in the most right. In the middle, which is the shortest path without elevation considering for reference and comparison. The leftmost figure is the database city map pre-loaded.

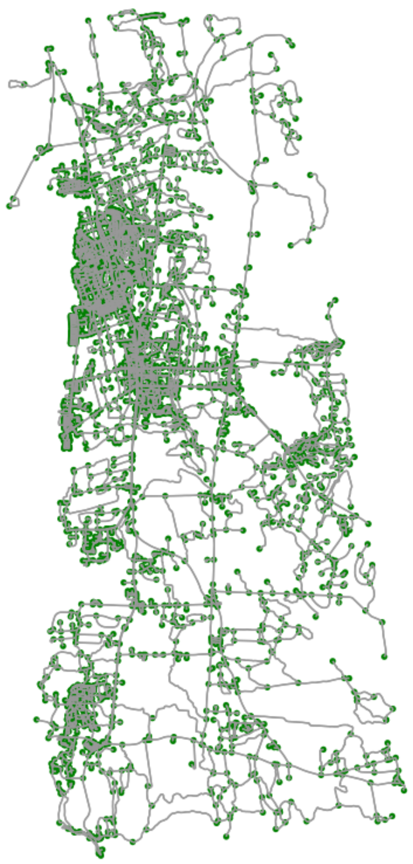
5. **Evaluation:**
   - Unit test: Test each component
     i. Test the data model
     ii. Test the input query each function
     iii. Test the navigation algorithms

     Compare our algorithms output with the output of the original Dijkstra algorithm without elevations which is the ground-truth
   - Integration test: Test the whole system offline with several outputs around a city, like Amherst.

In this evaluation we have tested the front end of our system, the backend data model and routing algorithms.

5.1 The front end test is tested by running each control on the interface. It shows it is working and also shown in the video demo too.

5.2 The data model
We have tested our data model with a "Amherst, MA" city model, the data graph is built with osmnx. The city plot is shown in the following Figure .



We have tested there are 5333 nodes and 15454 edges in the data graph. Each vertex has latitude, longitude, osmid, elevation information, such as {'y': 42.36905, 'x': -72.5133006, 'osmid': 6630490116, 'elevation': 75.833}). Each edge has length, street name and other information. Such as one edge has {'osmid': 173592470, 'name': 'Ray Stannard Baker Trail', 'highway': 'footway', 'oneway': False, 'length': 327.13300000000004, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f10999f6a58>, 'grade': 0.0555, 'grade_abs': 0.0555}). The length as weight is used only in our project.

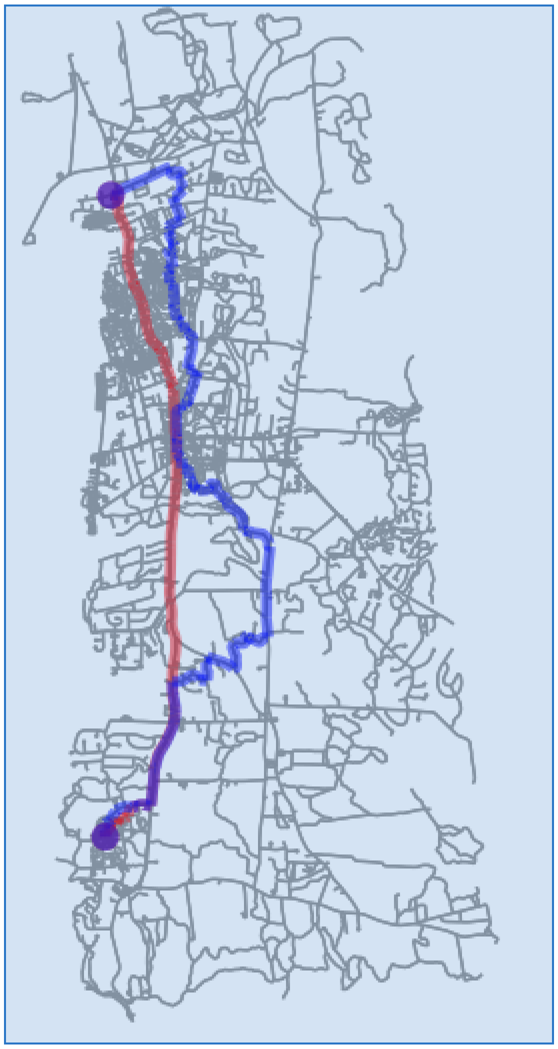
5.3. Routing algorithms.

We have tested our 4 different algorithms in different conditions, that is minimum elevation, Maximum elevation and 3 different percentages of shortest distance allowed, and compared with the ground truth shortest path algorithm(which does not consider elevation)

5.3.1 Comparison of ground truth with Dijkstra algorithm with min elevation

Here we show ground truth and the Dijkstra algorithm in minimum elevation with 50 percent.

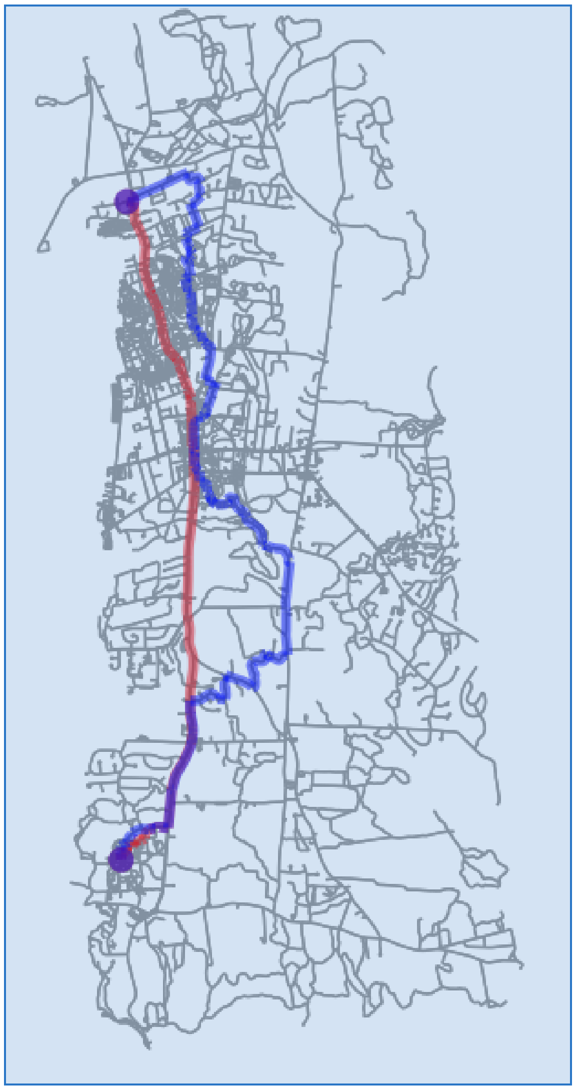
Given one source and destination, our algorithms generate the following results. The red color is the ground truth, the blue color is the Dijkstra algorithm. It shows it generate a different route.
The ground truth has the shortest distance: 9903.576999999994, the Dijkstra algorithm has the route distance: 14754.707000000006, which is 49 percentage of shortest path. It shows
It is working as expected



5.3.2 Comparison of ground truth with A* algorithm with min elevation

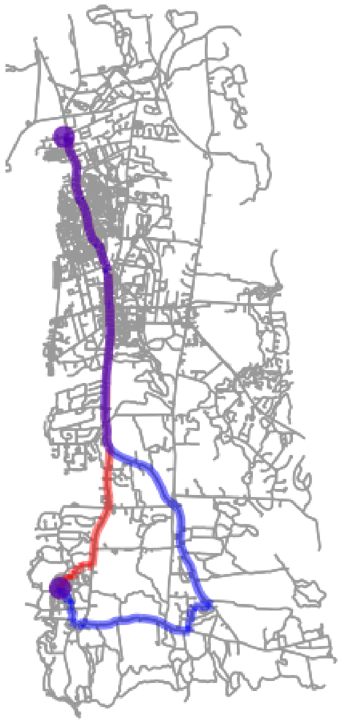
Here we show ground truth and the A* algorithm (with straight-line heuristc) in minimum elevation with 50 percent.
Given one source and destination, our algorithms generate the following results. The red color is the ground truth, the blue color is the Dijkstra algorithm. It shows it generate a different route.

The ground truth has the shortest distance: 9903.577, the A* algorithm has the route distance: 14754.707, which is 49 percentage of the shortest path. It shows it is working as expected. And also the routing returned by Dijkstra and A* the star algorithm are same.



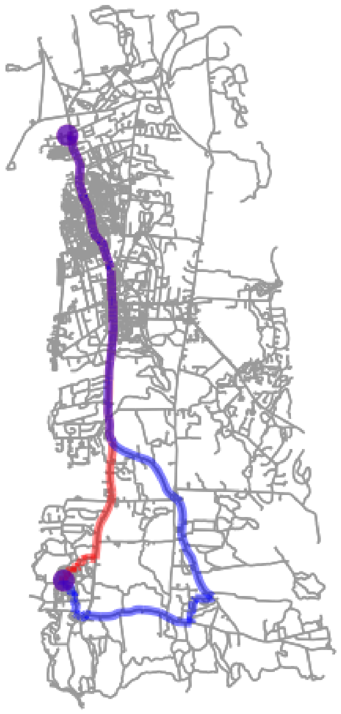
5.3.3 Comparison of ground truth with Dijkstra algorithm with max elevation

As shown in the figure below, the Dijkstra algorithm shows the max elevation with 50% percentage of shortest path



5.3.4 Comparison of ground truth with A* algorithm with max elevation

As shown in the figure below, the A* algorithm shows the max elevation with 50% percentage of shortest path, it shows the same path as the Dijkstra, which shows it works and can find the route.

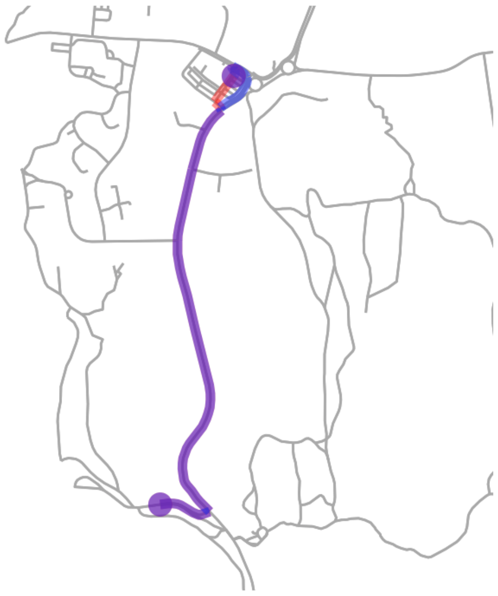


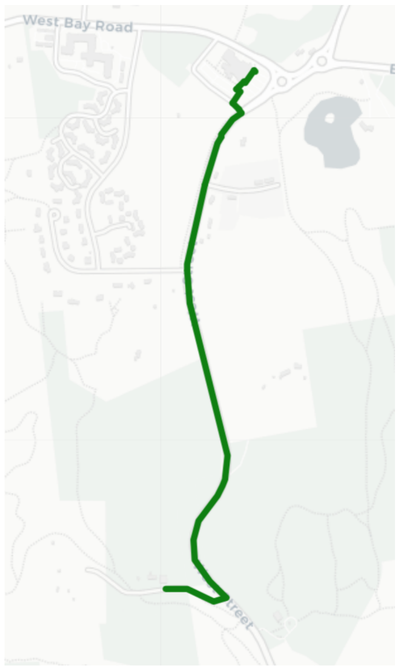
5.3.5 Comparison of ground truth with A* algorithm with max elevation

As shown in the figure below, the A* algorithm shows the max elevation with 50% percentage of shortest path, it shows the same path as the Dijkstra, which shows it works and can find the route.

5.3.6 Runtime comparisons among our algorithms and ground truth

BFS is really slow and it seems impossible to run BFS to get result as A* and Dijkstra as examples route above, but after adding some constraints that we will only search N-routes and return the best one we have. And relatively shorter routing. The right figure is our BFS best result in 200, and right is our Dijkstra result.





6. Related work:

Boeing, G., 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, *65*, pp.126-139.

McMinn, P., 2004. Search-based software test data generation: a survey. *Software testing, Verification and reliability*, *14*(2), pp.105-156.

Boeing, G., 2020. Urban Street Network Analysis in a Computational Notebook.