APPLICATION OF GRAPH THEORY

Term Project Final Report

NIU PATHFINDER

Shortest Distance between NIU Computer Science building and famous eateries and recreational spots in Dekalb

---

Team

Omer Bin Ali Bajubair - Z1905006
Shalini Reddy Challa - Z1931066
Abrar Athar Hashmi - Z1933888

---

1 Introduction

For this project our team has created a Graphical user Interface program to help students studying in NIU Computer Science building to find the fastest and shortest path between the source (Computer Science Building) to famous eateries and recreational spots in Dekalb

which is the destination target. The program uses an underlying simple weighted graph and utilizes vertex coloring to distinguish between intersections and buildings. When the user enters their schedule, the program executes Dijkstra's algorithm to determine the shortest path between buildings and highlights the path on the map of NIU and the eateries and recreational spots. We will be using dijkstra's algorithm and other graph related techniques to find the most efficient path. All locations are stored as nodes in the graph. Our algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph. Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. We will allow the user to enter the eateries and recreational spots that they want to visit in the GUI interface so that they can find the shortest and fastest path between the source and the target. Basically when the program is run the user will have a GUI in which he can select from the dropdown the place he wants to visit. Once that is selected then the algorithm runs in the background and the user will be provided with a pop up with the directions and estimated time and distance and also in the background a map will be displayed in the jupyter notebook and also that map will be saved in html file for the user.

---

2 Data Preparation / Summary of Data

Geopy is a python client for popular web services. For data collection we have used it to get the latitude and longitude to determine the distances between the NIU Computer Science building and famous eateries and recreational spots in Dekalb. With provided location, it is possible using Geopy to extract the coordinates meaning its latitude and longitude. Therefore, it can be used to express the location in terms of coordinates. These distances will be used as edge weights for the graph. We have extracted the latitude and longitude of our target places by running the Geopy script using python and stored them into an excel sheet. We utilized Google Maps' measurement tool allowing us to determine the distances between buildings and intersections. We also used Google Maps to obtain the coordinates of all intersections and buildings as well as the addresses and names of the buildings on the campus. More details of the data preparation is in the data excel sheet attached. The names which are empty on the 2nd column of the row (name on gmaps) are the exact same names that can be used as Places row. Hence it is kept empty. For the summary of the data we have taken some eateries and recreational spots in Dekalb. We have extracted the latitude and longitude for the places and calculated the distances between them which will act as the weights in the graph. Additionally for the construction of the graphs we have taken various points between the source and destination with their respective latitude and longitude that we have used in our algorithm. More details of those graph data will be attached for reference. When finding routes between two points, it is not very convenient to specify the latitude and longitude of the locations (unless you already have the coordinates in your dataset). Instead, it would be far easier to just specify their friendly names. You can actually perform this step (called geocoding) using the geopy module.Geocoding is the

process of converting an address into its coordinates. Reverse geocoding, on the other hand, turns a pair of coordinates into a friendly address.

PLACES	Name on gmaps	LATTITUDE	LONGITUDE
Psychology-Computer Science Building	Psychology-Computer Science Building	41.93162315	-88.76493668
BURGER KING	Burger king Dekalb	40.80899825	-73.95987847
HUSKIE DEN *		41.9358485	-88.76640956
QDOBA *		41.9317389	-88.7722459
STARBUCKS *	Starbucks Dekalb, USA	41.9317389	-88.7722459
CHIPOTLE	Chipotle, Dekalb, USA	41.9313782	-88.7716679
SUBWAY	Subway, Dekalb, USA	41.93055745	-88.7709822
TACO BELL		41.9312187	-88.77647305
DUNKIN		41.93135175	-88.7736953
POTBELLY		41.9313992	-88.7714973
VINNY'S PIZZA		41.931168	-88.7591379
PAPA JOHN		41.9310635	-88.75792869
JAMRAH DEKALB DOWNTOWN	Jamrah, Dekalb	41.9598333	-88.72001955
NIU REC CENTER			
AMC DEKALB	AMC Dekalb	41.94727645	-88.72056275
EGYPTIAN THEATER	EGYPTIAN THEATER, Dekalb	41.93120685	-88.75283369
ELLWOOD HOUSE MUSEUM	ELLWOOD HOUSE MUSEUM	41.93550345	-88.75347558
HSC	Holmes Student Center	41.9361137	-88.7671402
FOUR SEASON SPORTS		41.97212365	-88.719077
MUSIC BUILDING NIU	MUSIC BUILDING NIU, Dekalb	41.93548155	-88.76080194
SHABBONA LAKE		41.74531085	-88.85733441
YMCA	YMCA Kishwaukee Family, Dekalb	41.9651555	-88.72389508
JONAMAC Orchard	JONAMAC Orchard	41.9131394	-88.8645378

Figure 1: Overview of Data

3 Methods / Algorithm for Graph Model

Dijkstra's algorithm can be utilized to determine the fastest path. In this case, the algorithm begins at the first place the user selects, adds it to a shortest path tree list, then updates the weight of all the neighboring buildings with the edge weight plus the original node's weight. The program then advances to the next intersection vertex with the lowest weight and repeats.This continues until the target building is added to the shortest path tree list. Any vertices that are buildings other than the start and end building are ignored. Graphs are used to model connections between objects, people, or entities. They have two main elements: nodes and edges. Nodes represent objects and edges represent the connections between these objects. Dijkstra's Algorithm finds the shortest path between a given node (which is called the "source node") and all other nodes in a graph. This algorithm uses the weights of the edges to find the path that minimizes the total distance (weight) between the source node and all other nodes.

4 Working of the Algorithm

The dataset contains the latitudes and longitudes of source, destination and all the intermediate locations(i.e the streets). We used the latitude and longitudes to find the distance between each location. From the source i.e the NIU Computer Science building, the Djikstra algorithm finds the nearest neighbors and adds it to a list of visited nodes. An array stores the weights of the minimum distance from source to the current node. When each adjacent node is visited, we check for the minimum of current weight and previous weight + current distance, and the minimum of both is selected. We have two dictionaries, shortest-path and previous-nodes: shortest-path will store the minimum cost of visiting each city in the graph starting from the start-node. In the beginning, the cost is infinity for all nodes, but we'll update the values as we move along the graph. previous-nodes will store the path of the current best known path for each node. For example, if we know the best way to 3 to be via 2, previous-nodes["3"] will return "2", and previous-nodes["2"] will return "7" i.e the final target. We'll use this dictionary to backtrace the shortest path. The algorithm visits all node's neighbors that are still unvisited. If the new path to the neighbor is better than the current best path, the algorithm makes adjustments in the shortest-path and previous-nodes dictionaries. Lastly, we need to create a function that prints out the results. This function will take the two dictionaries returned by the dijskstra-algorithm function, as well as the names of the beginning and target nodes. It'll use the two dictionaries to find the best path and calculate the path's score.

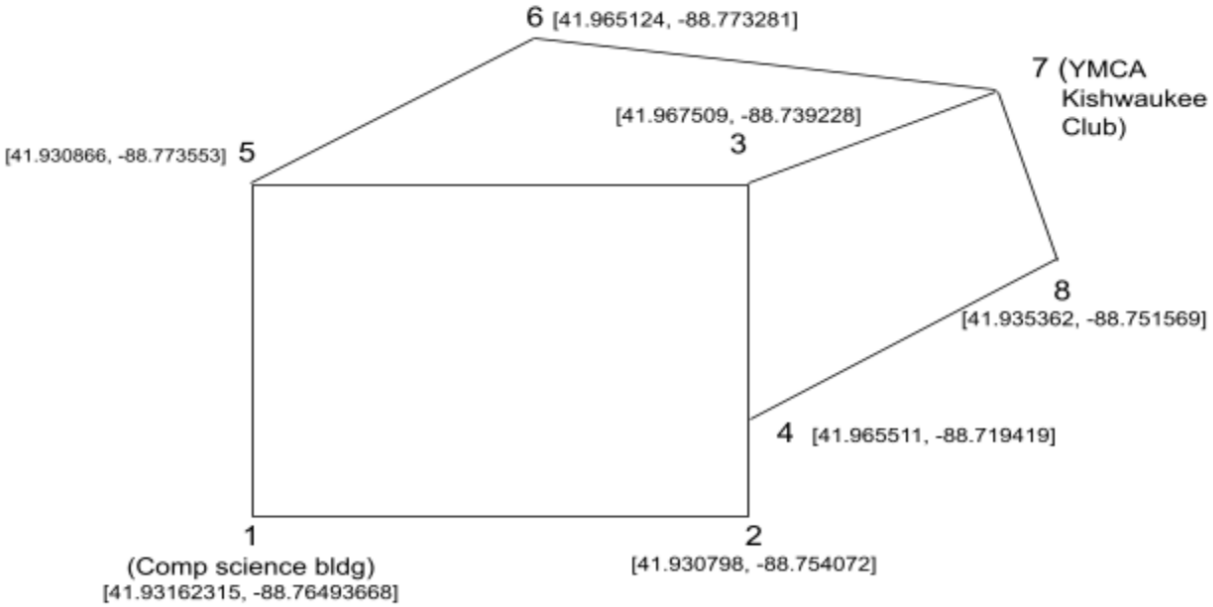

Figure 2: Graph Construction Edge Intersection Map

5 Output GUI

We have designed a GUI which has a dropdown where the user has the choice to select his destination spot. Once the user selects the destination and the source is by default the NIU Computer Science building. Once this is done a pop up box with the directions along with estimated arrival time and estimated distance in miles is presented on the screen for the user to navigate and also more importantly a map is saved locally which shows the source and target with the map. Also we have provided various map options for the user like if he wants to see the openstreetmap view, stamen terrain, stamen toner etc views to the user. OSMnx is a Python package that lets you download geospatial data from OpenStreetMap and model, project, visualize, and analyze real-world street networks and any other geospatial geometries.

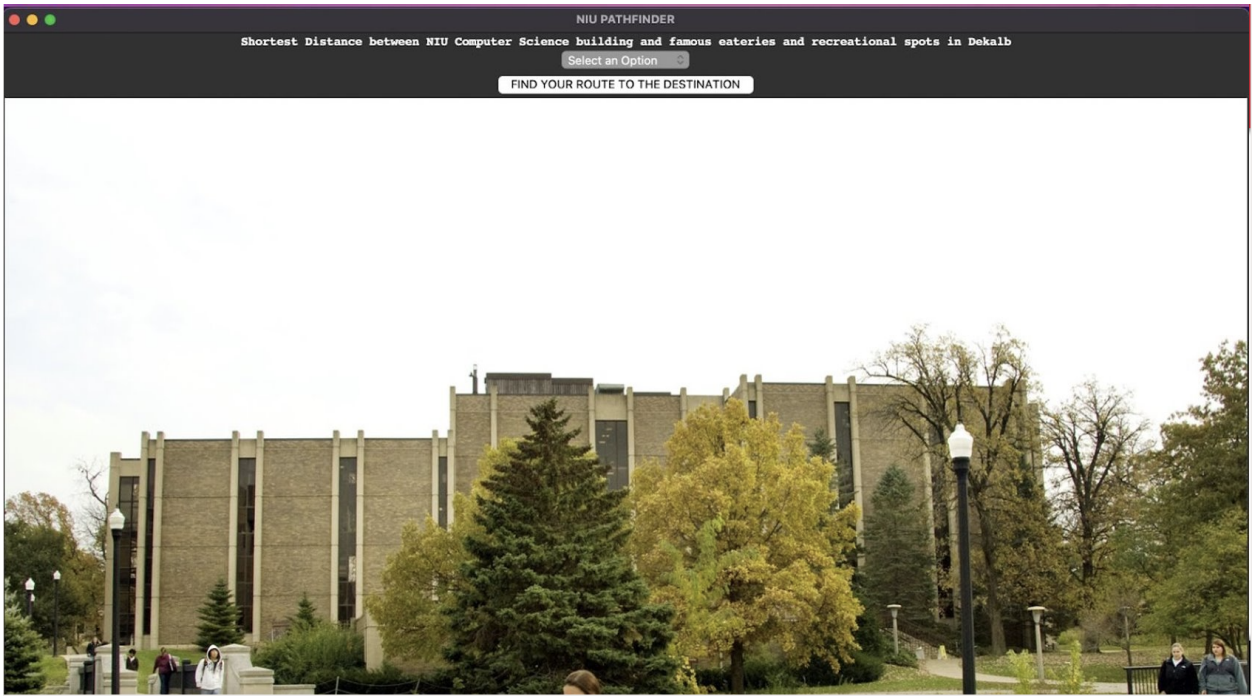

Figure 3: GUI Overview

As you can see how our GUI looks . It has a header titled as NIU PATHFINDER and a tag line. Select an option is a dropdown from which the user can select different places he can visit. Find your route to the destination is a button which clicked by the user in the backend runs the algorithms and provides the results.

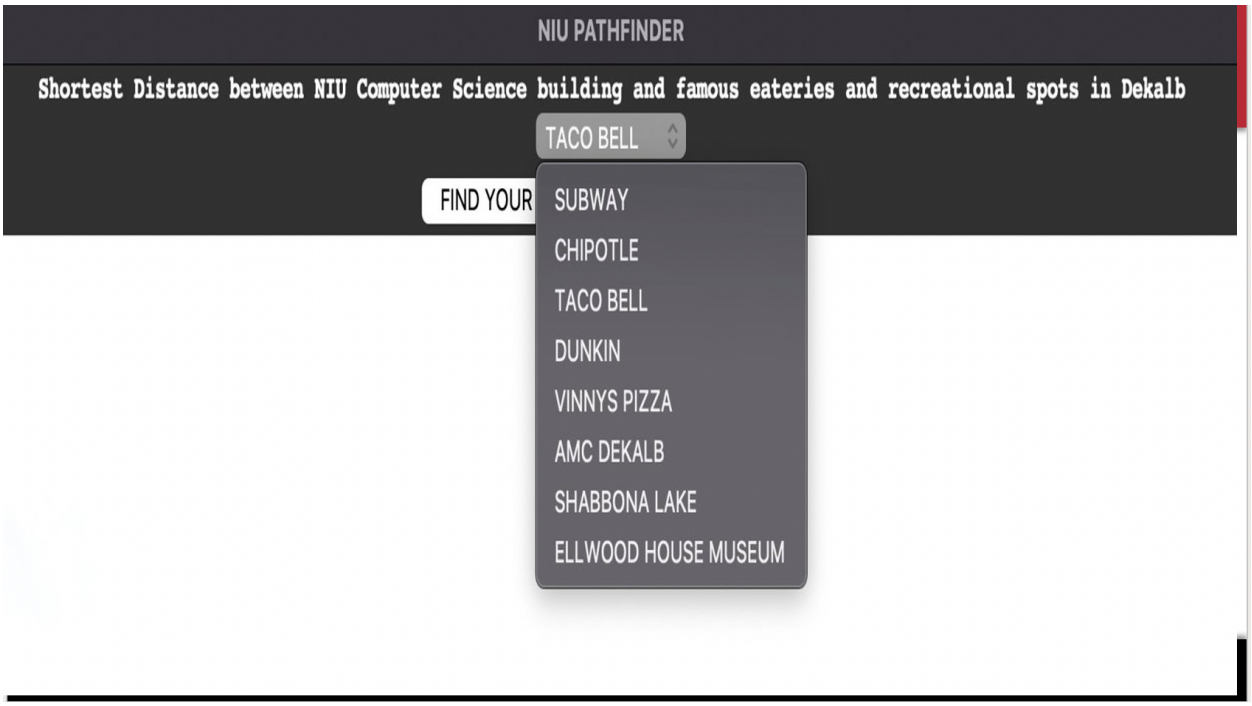

Figure 4: GUI Location Dropdown List

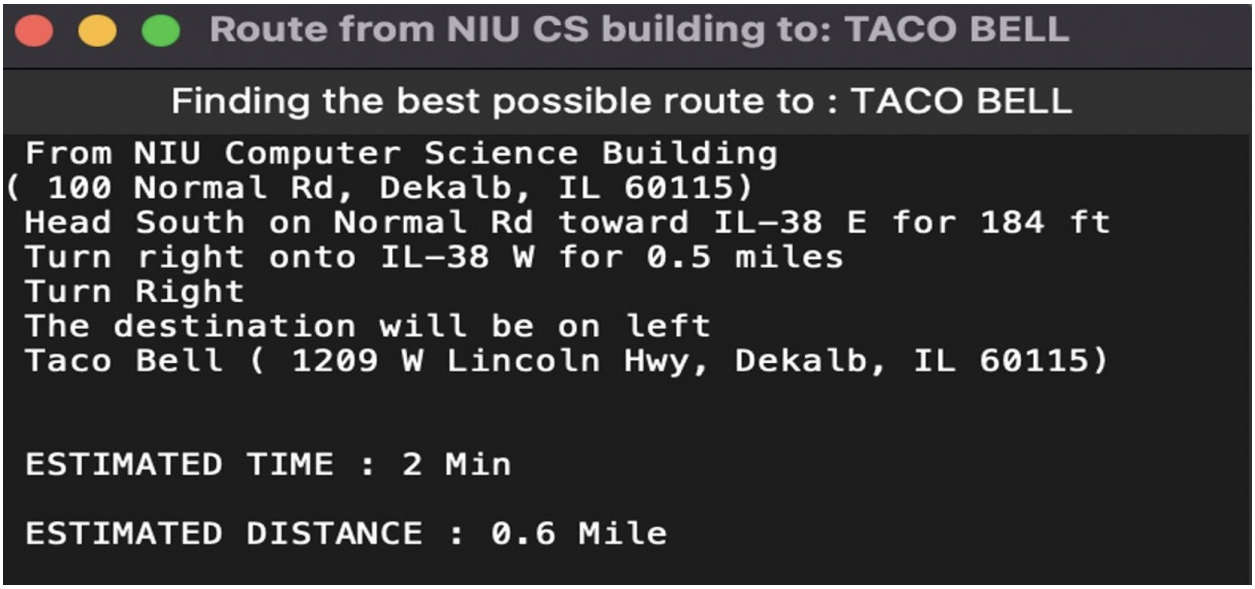

Figure 5: Directions / Estimated Time / Distance Pop up

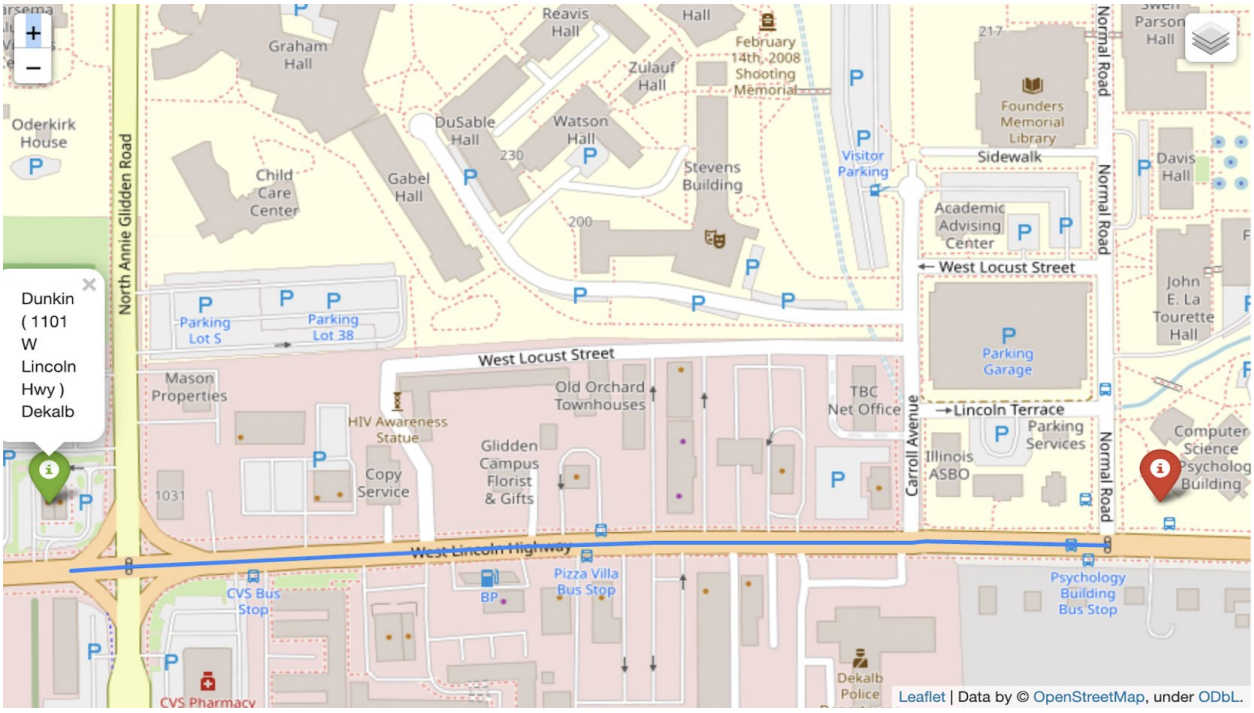

Figure 6: Output Map showing path between Source and Destination

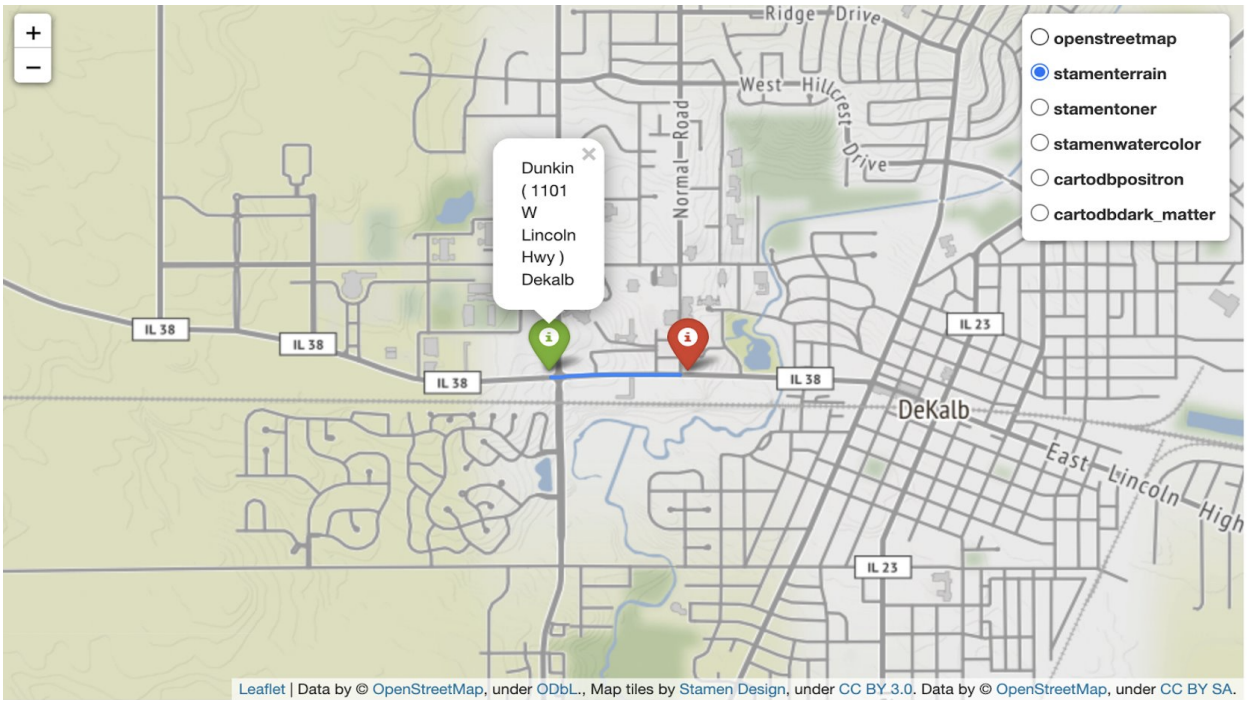

Figure 7: Various forms to view the Map

6 Results

* Pop up with directions, estimated time and estimated distance. * Map is saved locally for the user to view in .html * Output map with various backgrounds in notebook with path

highlighted and source and destination as markers.

We obtained comparative results for the project by using a static destination list dropdown and running it through Google Maps and our program, and then comparing the results.

Comparing with Google maps our working example GUI: Suppose if you select you want to go to shabbona lake. You select that on the GUI. A pop up will show the directions with the shortest path based on miles.

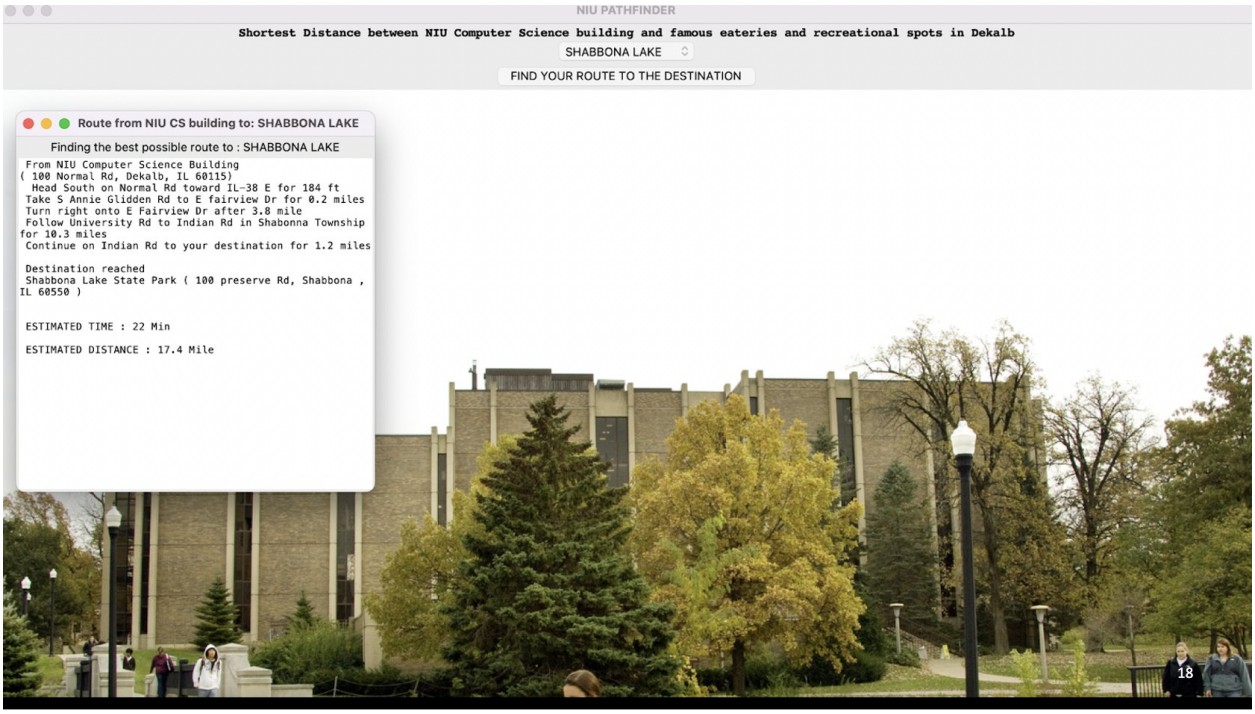

Figure 8: Output GUI

As you can see below the picture is taken from google maps. So the shortest path is 28 mins and that is 19.2 miles. We have constructed a graph of nodes with various points as shown below with latitude and longitude of the positions. From the source i.e the NIU Computer Science building, the Djikstra algorithm finds the nearest neighbors and adds it to a list of visited nodes. An array stores the weights of the minimum distance from source to the current node. When each adjacent node is visited, we check for the minimum of current weight and previous weight + current distance, and the minimum of both is selected. We have two dictionaries, shortest-path and previous-nodes: shortest-path will store the minimum cost of visiting each city in the graph starting from the start-node. In the beginning, the cost is infinity for all nodes, but we'll update the values as we move along the graph. previous-nodes will store the path of the current best known path for each node. The algorithm visits all node's neighbors that are still unvisited. If the new path to the neighbor is better than the current best path, the algorithm makes adjustments in the shortest-path and previous-nodes dictionaries.

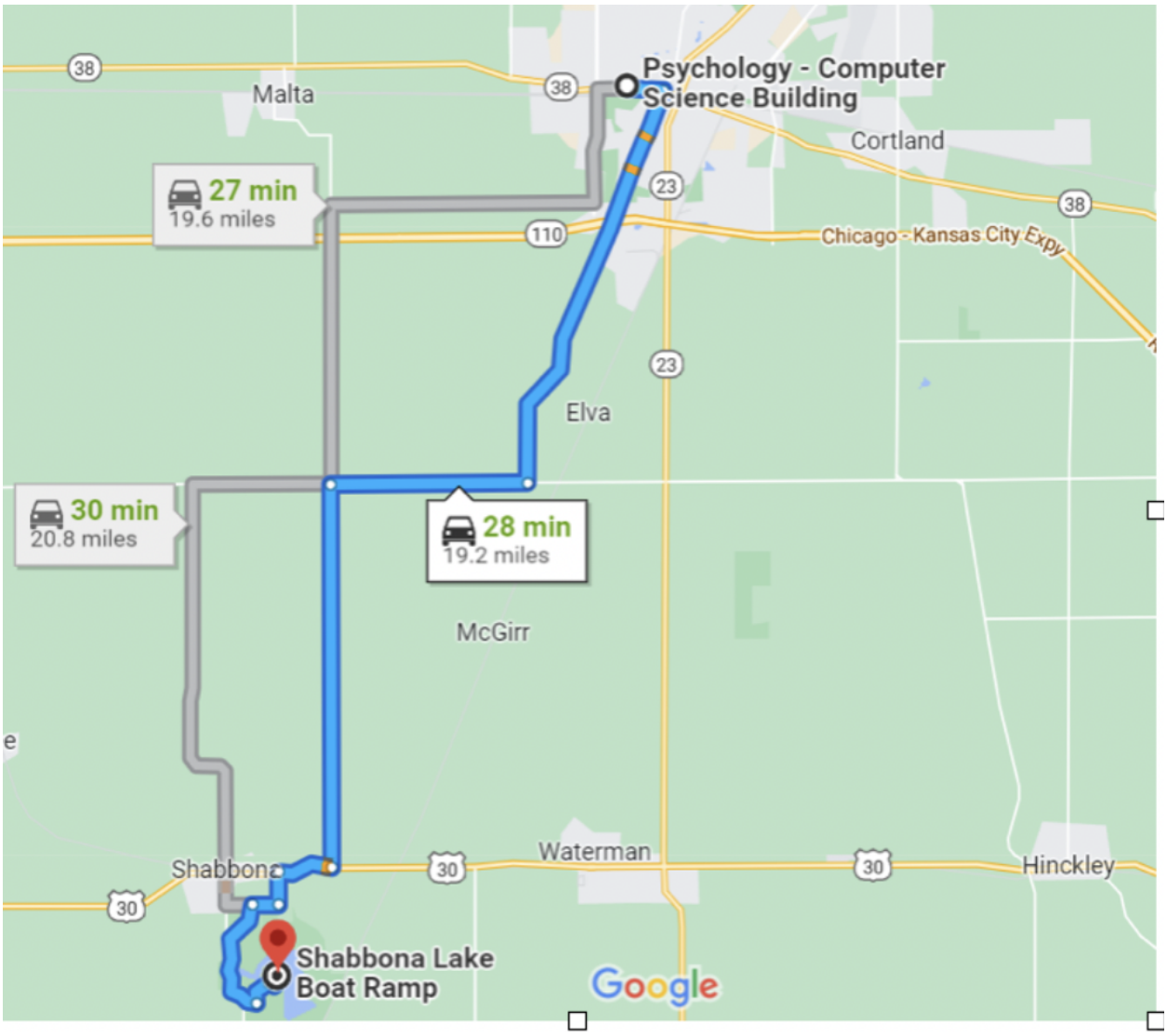

Figure 9: Google Map representation

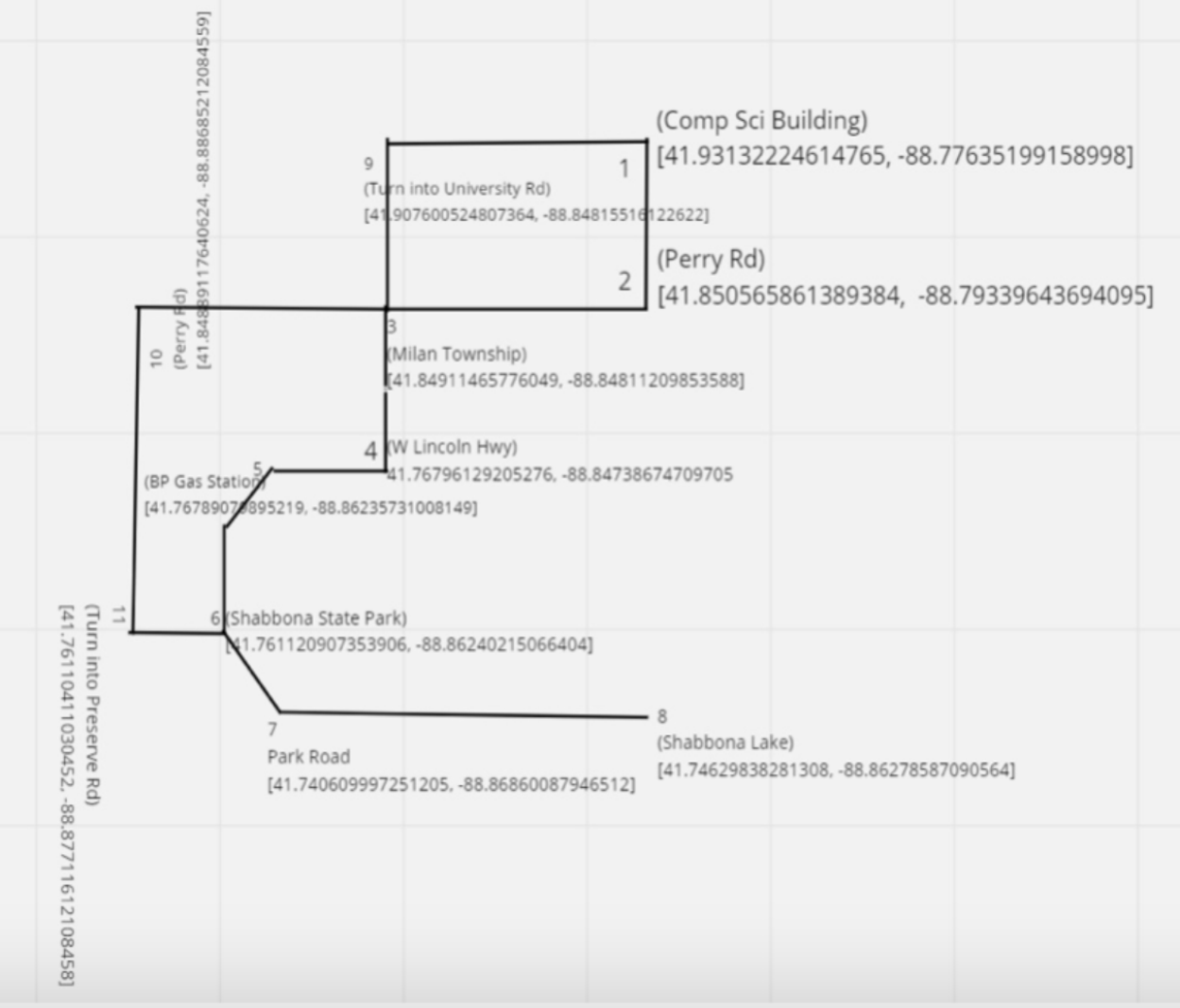

Figure 10: Our Graph Construction with edges

Shabbona Lake

1. [41.93132224614765, -88.77635199158998]
2. [41.85056861389384, -88.79339643694095]
3. [41.84911465776049, -88.84811209853588]
4. [41.76796129205276, -88.84738674709705]
5. [41.76789079895219, -88.86235731008149]
6. [41.76112090735390 6, -88.86240215066404]
7. [41.74060997251205, -88.86860087946512]
8. [41.74629838281308, -88.86278587090564]
9. [41.90760052480736 4, -88.84815516122622]
10. [41.84889117640624, -88.88685212084559]
11. [41.76110411030452, -88.87711612108458]

Figure 11: Our Graph Node Points

As you can see below this is the output that we are getting from our algorithm. As you can compare with the google maps picture mentioned above and our result below we are able to match the shortest path results through our implementation. Also we can view our output map in various backgrounds. The green marker on the map is the destination.

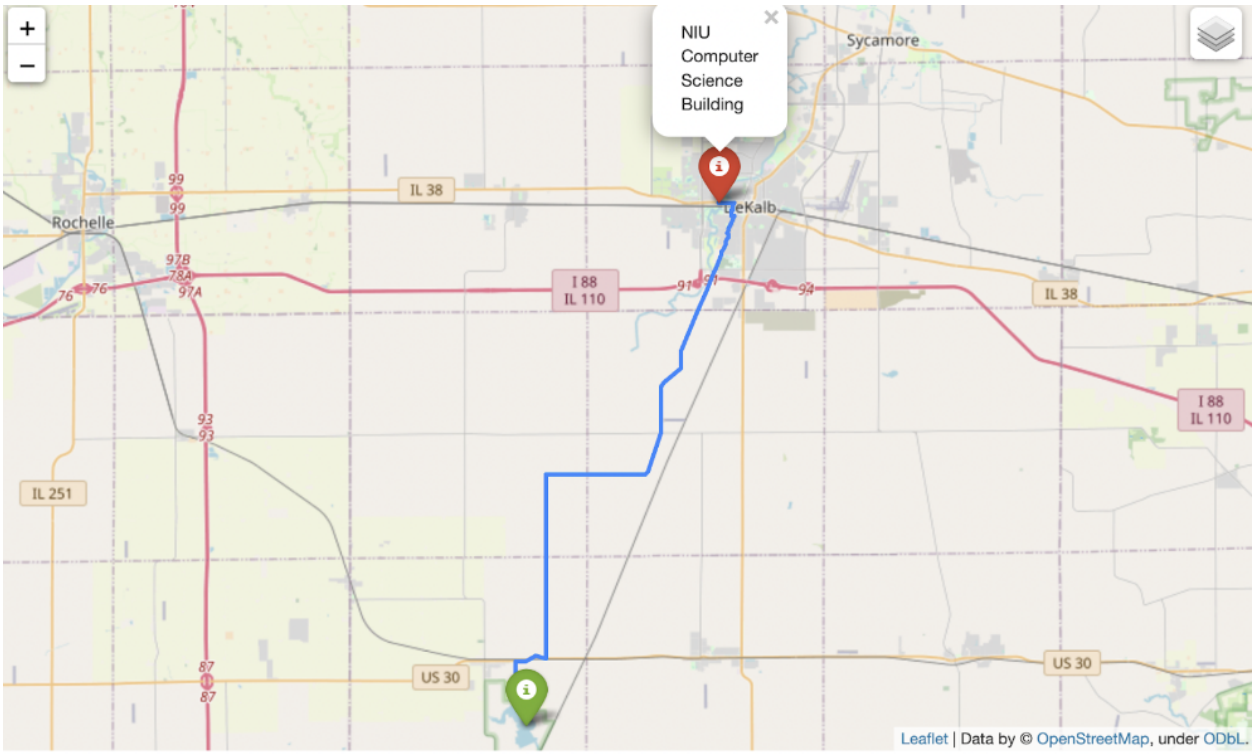

Figure 12: Our Map Output

---

7 Programming Language System

Python / GeoPy / Google Map API / Gmplot / Tkinter (For GUI) / VSCODE / mac / windows (compatible for both) / Folium for maps / OSMnx (openstreetmap) / Jupyter notebook

We have used python for coding our project and implementing the algorithm. We have used Geopy / Google Map API to fetch the latitude and longitude of the places for data preparation. Tkinter was used to make the graphical user interface for our project. Gmplot / folium / OSMnx packages were used for implementing the map visualizations. All the coding was done in a jupyter notebook and visual studio code IDE.

---

8 Discussion

While our method did produce better results than Google Maps, it was not perfect. The largest issue with our program comes from the method by which distances and coordinates were collected from Google Maps. Because distances were manually collected from Google Maps, some precision in the path length was lost.

---

9 Future Work

Expanding the Graph : More Destination spots Due to time limitations, the graph only includes a few places from Dekalb. Expanding the graph to reach this area would allow us to improve navigation for our users.

Full Map Path Outlining : Due to time restraints, we were unable to add path outlining to all paths on the map. Our map displays only the shortest path. With more time we would have fixed this.

Expanding Functionality : Adapting for Mobile Now that we know our algorithm is successful, it would be very helpful to users if we could adapt it to Android or iOS devices to assist in navigation. This would also enable GPS navigation allowing users to receive directions in real-time.

---

10 Github Repo for code and documentation

Link : https://github.com/omer1997/GRAPH-THEORY

---

11 Acknowledgments

We would like to thank Dr. Minmei Hou for giving us the opportunity to do this project. Without the assistance and dedicated involvement in every step throughout the process, this project would have never been accomplished.
