Navigation System Based on Graph Theory

Yuchen Xia
School of Information Science and Engineering,
Shandong University, Qingdao 266237, China.
Emails: yuchenxia@mail.sdu.edu.cn

***Abstract*—Navigation systems have become an essential part of our daily lives, helping us to navigate through unfamiliar territories and reach our destinations efficiently. One of the key components of navigation systems is the use of graph theory, which allows for the representation of maps and routes as graphs with nodes and edges. This paper focuses on the development of a navigation system based on graph theory, which uses Dijkstra's algorithm. You can find the relevant code here: https://github.com/MonsterXia/Navigation-System-Based-on-Graph-Theory**

***Index Terms*—Navigation systems, Graph Theory, Dijkstra's algorithm**

I. Introduction

Navigation systems have become an integral part of our daily lives, providing us with accurate and efficient routes to our destinations. These systems rely on complex algorithms and data structures to handle large amounts of information and compute the best possible routes. Graph theory, which is the study of graphs and their properties, has emerged as a powerful tool for modeling and solving problems in a wide range of fields, including navigation systems.

In this paper, we present a navigation system based on graph theory, rebuilt by Dijkstra's algorithm [1]. Dijkstra's algorithm is a widely used algorithm for finding the shortest path between two nodes in a graph with non-negative edge weights. It is a greedy algorithm that iteratively selects the node with the smallest distance from the source node and updates the distances of its neighbors. This algorithm has been widely used in navigation systems due to its simplicity and efficiency.

Our navigation system is designed to provide users with the shortest path between two points on a map, using Dijkstra's algorithm. The system takes in user input for the starting and ending points, and then constructs a graph representation of the map. The nodes of the graph represent the locations on the map, and the edges represent the roads that connect them. The weights of the edges represent the distance between the locations, and are computed using real-world data such as road length and speed limits.

Once the graph representation is constructed, Dijkstra's algorithm is used to compute the shortest path between the starting and ending points. The algorithm iteratively selects the node withthe smallest distance from the starting point and updates the distances of its neighbors. This process continues until the shortest path to the destination is found. The shortest path is then presented to the user in the form of a map, along with turn-by-turn instructions for reaching the destination.

One of the key advantages of using Dijkstra's algorithm in our navigation system is its ability to handle complex road networks. The algorithm can efficiently compute the shortest path even when there are multiple routes to the destination or when there are traffic restrictions on certain roads. Additionally, the algorithm can be easily modified to incorporate real-time traffic data, enabling our system to provide more accurate and up-to-date routes to users.

In the second part of this paper, we will provide a detailed exposition of Dijkstra's algorithm. In the third part, we will propose specific solutions to concrete problems based on Dijkstra's algorithm.

II. Dijkstra's algorithm

Dijkstra's algorithm is a well-known graph search algorithm that is used to find the shortest path between a starting node and all other nodes in a weighted graph. It was developed by Dutch computer scientist Edsger W. Dijkstra in 1956 and has since become an essential algorithm in computer science.

The algorithm begins at the starting node and considers all neighboring nodes, calculating the distance from the starting node to each of these nodes. It then chooses the neighboring node with the smallest distance and adds it to a set of visited nodes. The algorithm then repeats this process, considering the unvisited neighboring nodes of the nodes that have already been visited until all nodes in the graph have been visited.

Dijkstra's algorithm requires that the graph be represented as an adjacency matrix or an adjacency list. The adjacency matrix represents the graph as a two-dimensional array, where each row and column correspond to a node in the graph, and the value at the intersection of the row and column represents the weight of the edge between the two nodes. The adjacency list represents the graph as a list of nodes, where each node is associated with a list of its neighboring nodes and the weight of the edges between them.

The algorithm maintains two data structures: a set of visited nodes and a set of unvisited nodes. Initially, all nodes are unvisited, except for the starting node, which is added to the set of visited nodes. The algorithm also maintains a distance array, which stores the distance from the starting node to each nodein the graph. Initially, the distance to the starting node is set to 0, and the distance to all other nodes is set to infinity.

The algorithm then iterates through the unvisited neighboring nodes of the starting node, calculating the distance from the starting node to each of these nodes based on the weights of the edges between them. It updates the distance array accordingly, setting the distance to each neighboring

node to the sum of the distance to the current node and the weight of the edge between them.

The algorithm then chooses the neighboring node with the smallest distance and adds it to the set of visited nodes. It repeats this process, considering the unvisited neighboring nodes of the nodes that have already been visited, until all nodes in the graph have been visited.

At the end of the algorithm, the distance array holds the shortest distance from the starting node to all other nodes in the graph. It can also be used to reconstruct the shortest path from the starting node to any other node in the graph.

Specifically, the implementation steps of the algorithm are as follows:

1) Add the source node to the set S and set $dis[s]$ to 0, where s is the source node.
2) For each neighboring node v of the source node, update the value of $dis[v]$ to $dis[s]+$ the weight of the edge (s, v).
3) Choose the node u with the smallest dis value from the dis array, which is not yet in set S.
4) Add node u to set S and update the dis values of its neighboring nodes. If the updated dis value is smaller than the original value, update the dis array.
5) Repeat steps 3 and 4 until all nodes have been added to set S.

Dijkstra's algorithm has a time complexity of $O(|E| + |V| \log |V|)$, where $|E|$ is the number of edges in the graph and $|V|$ is the number of nodes in the graph. This makes it an efficient algorithm for finding the shortest path in small to medium-sized graphs. However, it may not be suitable for very large graphs, as the time complexity can become prohibitive.

In conclusion, Dijkstra's algorithm is a powerful tool for finding theshortest path between a starting node and all other nodes in a weighted graph. It is based on a simple idea of visiting neighboring nodes in order of their distance from the starting node, updating the distance array as needed. Despite its simplicity, it is a very effective algorithm that has many real-world applications, such as in routing protocols for computer networks, transportation systems, and logistics planning.

The algorithm has several variations, such as the A* algorithm [2], which is an extension of Dijkstra's algorithm that uses heuristics to guide the search towards the goal node, and the bidirectional Dijkstra's algorithm, which searches from both the starting node and the destination node simultaneously, reducing the search space and thus improving the efficiency.

Dijkstra's algorithm is not without its limitations, however. It assumes that the graph is connected and that all edges have non-negative weights. It also does not work well with graphs that have negative cycles, as it can get stuck in an infinite loop. To handle these cases, other algorithms such as the Bellman-Ford algorithm [3] and the Floyd-Warshall algorithm [4] are used.

III. Navigation systems

When we open the map, we will find a complex network of roads distributed between cities, as shown in Figure 1. Taking Shandong Province as an example, intercity highways, expressways, national highways, and county roads connect the 16 prefecture-level cities. If we want to create a detailed model, we need to use each road intersection as a vertex and each road as an edge. We also need a vast amount of detailed data to support the model, including information about the road's traffic conditions and length, which will be used to set the weight of each edge. Therefore, in order to illustrate the principles more clearly, we simplified the modeling process.

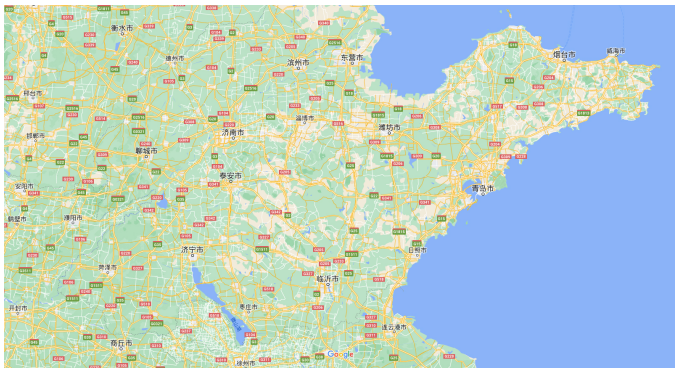

Fig. 1. Actual Map

We took the 16 prefecture-level cities as the vertices, and if there is a direct expressway between two cities without detours, we consider that there is an edge between the two vertices representing the cities. Since the selected edges represent expressways, and the road conditions are relatively consistent, the weight of each edge is roughly proportional to the straight-line distance between the two vertices representing the cities. In this case, we can abstract the transportation map of Shandong Province as the graph represented in Figure 2.

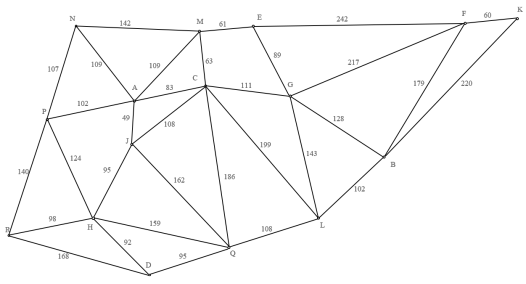

Fig. 2. MODEL

In Figure 2, we use letters to represent the cities. The correspondence between the letters and the cities is shown in Table I. The weight of each edge represents the straight-line distance between the two cities connected by the edge, and the unit is kilometers.

In the implementation of the graph, we use an adjacency matrix [5] to represent the relationship between the graph and its vertices.

Based on the principles described in the second part called "Dijkstra's algorithm", I implemented a shortest path algorithm model after abstracting and modeling the graph as mentioned above.

Assuming we choose Jinan as our starting point and calculate the shortest paths to all other cities in Shandong Province, we obtain the data shown in Table I.

TABLE I
THE SHORTEST DISTANCE FROM JINAN TO OTHER CITIES

Destination	Code	Distance/km	Path
Jinan	A	0	@
Qingdao	B	322	L
Zibo	C	83	J
Zaozhuang	D	236	R
Dongying	E	170	G
Yantai	F	411	B
Weifang	G	194	C
Jining	H	144	P
Taian	J	49	A
Weihai	K	471	F
Rizhao	L	282	G
Binzhou	M	109	N
Dezhou	N	109	P
Liaocheng	P	102	A
Linyi	Q	211	D
Heze	R	242	H

The @ in the table indicates that the departure and destination are the same, thus there is no previous path point.

The first column of the table represents the destination, or the endpoint of our path. The second column contains the codes corresponding to each city, replacing the full names of the cities for convenience. The third column shows the shortest path required to reach the destination, while the fourth column indicates the previous node on the path to the endpoint. With the data in the fourth column, we can obtain all the complete shortest paths to each destination.

Taking the example of starting from Jinan (A) and ending at Qingdao (B), the previous node on the path to B is G, and the previous node on the path to G is C, and so on. Therefore, we obtain the shortest path from A to B as $A \rightarrow C \rightarrow G \rightarrow B$, with a length of 322 kilometers. This route has been verified to be correct through navigation software and visual observation on the map. This verifies that Dijkstra's algorithm can quickly find the shortest path and highlights the possibility of simplifying graph modeling in specific environments.

IV. CONCLUSION

In conclusion, our navigation system based on graph theory and rebuilt by Dijkstra's algorithm provides a powerful and efficient tool for navigating through complex road networks. It enables users to quickly and easily find the shortest path between two points on a map, and provides turn-by-turn instructions for reaching the destination. The use of Dijkstra's algorithm allows the system to handle complex road networks and incorporate real-time traffic data, making it a valuable tool for both daily commuting and long-distance travel.

REFERENCES

[1] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in Edsger Wybe Dijkstra: His Life,Work, and Legacy, vol. 45: Association for Computing Machinery, 2022, pp. 287–290.

[2] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

[3] R. Bellman, "On a routing problem," Quarterly of applied mathematics, vol. 16, no. 1, pp. 87-90, 1958.

[4] J. Munkres, "Algorithms for the assignment and transportation problems," Journal of the society for industrial and applied mathematics, vol. 5, no. 1, pp. 32-38, 1957.

[5] G. Chiaselotti, T. Gentile, F. G. Infusino, and P. A. Oliverio, "The adjacency matrix of a graph as a data table: a geometric perspective," Annali di Matematica Pura ed Applicata (1923 -), vol. 196, no. 3, pp. 1073-1112, 2017/06/01 2017, doi: 10.1007/s10231-016-0608-1.