Travelling Salesman Problem

Noran Mohamed , Kanza Haider , Seyma Altay
Istanbul Sehir University
34865 Dragos, Istanbul, Turkey
{noranmohamed,kanzahaider,seymaaltay}@std.sehir.edu.tr

Abstract

In this report, a network of a hiking trail map famously known as the Sleeping Giant park located in the town of Hamden in Connecticut, United States, is explored. The Travelling Salesman Problem(TSP) technique is applied on the data set of the Sleeping Giant hiking trail route map consisting of edges(trails) and nodes(objects) to find the best possible strategy for a hiker to move from node to node forming a minimum-cost Eulerian tour of the computed graph. The dataset is also analyzed to find the higher priority nodes using the centrality measures of high order degrees and betweenness.

1 Introduction

Many problems in the real world are related to networks or graphs. The Travelling Salesman Problem (TSP) is an arc routing problem which is a mathematical problem of graph theory itself. It is an NP-hard problem in combinatorial optimization which makes it important in operations research and theoretical computer science. Its application is not just limited to road distance networks but it extends as far as several other areas in its purest formulations such as planning and logistics.[1]

The objective of the TSP is to find the shortest path that covers all the links (roads) on a graph at least once. Given a graph G, the TSP is to find a minimum length Eulerian tour traversing all arcs of G at least once. If this is possible without doubling back on the same road twice which is an ideal scenario and makes the problem in hand quite simple. However, if some roads must be traversed more than once, there comes the need of some math to find the shortest route that hits every road at least once with the lowest total mileage. This is where the Dijkstras theorem shall be used which is explained later in the report.[2] We implemented the TSP algorithm to solve the problem on a real graph (trail network of a state park) using the NetworkX library in Python.

1.1 Motivation

Our personal motivation to choose for the Sleeping Giant Park hiking data to solve the problem for examining the shortest route came for apparent two main reasons:

i. First, the sleeping giant park itself consists of a very interesting topography that makes it famous for what it is called. As the name speaks, the profile features distinct head, chin, chest, hip, knee and feet topographically through crops and ridges. This fact makes it very interesting topic to study.

ii. Secondly, the hikers or tourist often with a motivation to adventure all the trails in the park wonders repeating the same trails multiple times and so fail to explore all the trails. Hence, solving this problem using a TSP can provide the best strategy for hikers to move from trails after trail till they do a complete tour of the park with least repetition of the joining tracks.

2 Data

The data comprises an edge-weighted undirected or a bidirectional graph where the edges do not have any orientation. That is to say in order to hike the trail could be, either way, for example, A<−−−>B == B<−−−>A. Fig 1. Shows the graph of the Sleeping giant park took from the sleeping giant park association website www.spga.org.The dataset of the sleeping giant consists of two sets of data i.e. edge list and node list.

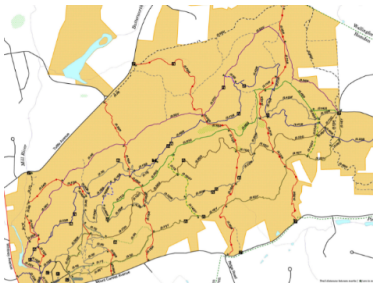

Figure 1: Sleeping Giant Park Map.

2.1 Edge List

Edge list is a simple data structure consisting of five columns specifying node1,node2, trail, distance, color, and estimate. Node1 and node2 are the nodes that are connected. The trail is the abbreviated name of each trail. Distance indicates the trail length in miles. Color is the trail color use for plotting. Estimate tells whether the edge distance is an approximation of the trail map (1=yes, 0=no) as some distances are not provided. Table 1 enlists the first 10 data points.

	node1	node2	trail	distance	color	estimate
0	rs_end_north	v_rs	rs	0.30	red	0
1	v_rs	b_rs	rs	0.21	red	0
2	b_rs	g_rs	rs	0.11	red	0
3	g_rs	w_rs	rs	0.18	red	0
4	w_rs	o_rs	rs	0.21	red	0
5	o_rs	y_rs	rs	0.12	red	0
6	y_rs	rs_end_south	rs	0.39	red	0
7	rc_end_north	v_rc	rc	0.70	red	0
8	v_rc	b_rc	rc	0.04	red	0
9	b_rc	g_rc	rc	0.15	red	0

Table 1 : Edge list Data.

2.2 Node List

X, Y coordinates of the nodes (trail intersections) are used so that we can plot the graph with the same layout as the trail map. Table 2 illustrates the first five data points.

	id	X	Y
0	b_bv	1486	732
1	b_bw	716	1357
2	b_end_east	3164	1111
3	b_end_west	141	1938
4	b_g	1725	771

Table 2: Node List.

3 Solution Methodology

We found that NetworkX package in python has the strongest graph algorithms which were needed to solve the TSP on our data of hiking trail network. Hence, we used NetworkX to implement steps of TSP.

3.1 Overview of TSP Algorithm

TSP Step 1: Plot the graph of the original map
TSP Step 2: Find Min Distance Node Pairs
- Step 2.1: Find nodes of odd degree
- Step 2.2: Compute odd degree node pairs
- Step 2.3: Find the shortest path between the odd degree nodes (Apply Dijkstras Theorem)
- Step 2.4: Compute Minimum Weight Matching
- Step 2.5: Augment the Original Graph

TSP Step 3: Compute Eulerian Circuit

3.1 Plotting the original map

Figure 2 shows the plot of original sleeping giant map created.

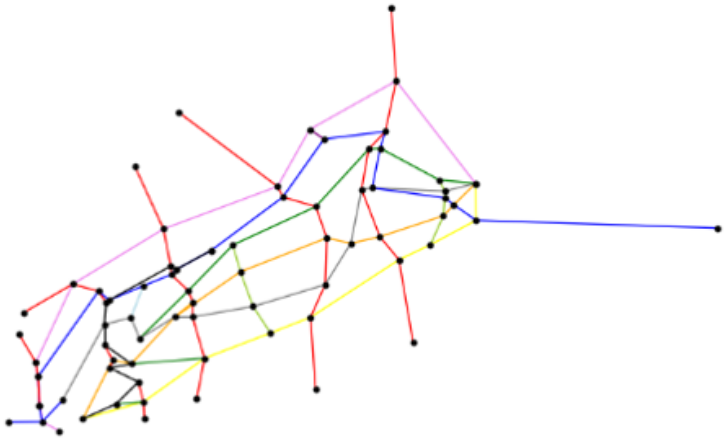

Figure 2: Created Map

3.2.1 Find Nodes of Odd Degree

As part of the general rule of even and odd degree graphs, we know that a graph comprising of an odd degree edges means a graph with dead endpoints. In such cases, it is not possible to visit all of the nodes in the graph without repeating any of the edges twice. Hence, it was essential as the first step to find how many odd degree nodes did our hiking trail data consisted. It was found that out of 123 nodes, 36 nodes were odd degree nodes.

3.2.2 Compute odd degree node pairs

36 odd node pairs makes a combination pair of 630 edges. Fig 3 shows the plot whereby 36 odd node pairs are connected with each other through 630 edges.

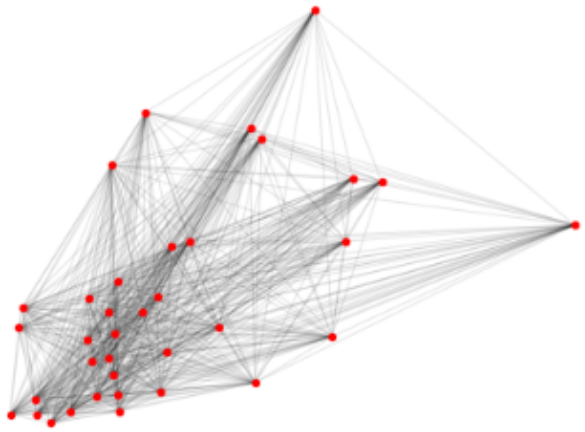

Figure 3: 36 odd node with 630 connecting edges.

3.2.3: Finding shortest path between odd nodes

Here we will apply the Dijkstras Theorem to compute the minimum distance among the odd node pairs. NetworkX provides a convenient function dijkstra_path_length that returns the shortest path length from source to target in a weighted graph.

3.2.4: Compute Minimum Weight Matching

Minimum Weight Matching is a special function with NetworkX package that in actual has almost 800+ lines of code. It is used to capture the minimum distance edge from every node and retain into the graph.

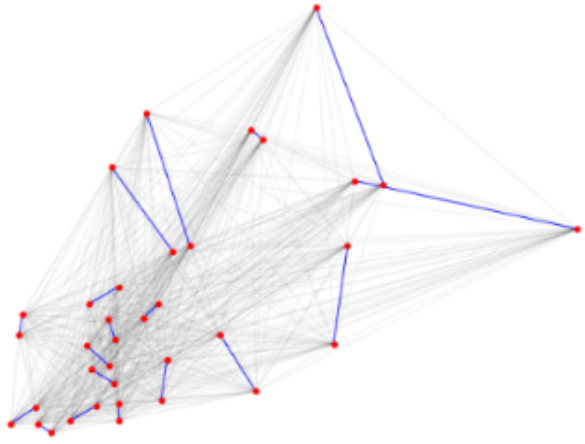

Figure 4: Minimum weight matching on complete graph.

3.2.5: Augment the Original Graph

We shall then augment the minimum weight matching graph into the original graph plotted in Step 1. Fig 5 below shows the augmented graph plotted.

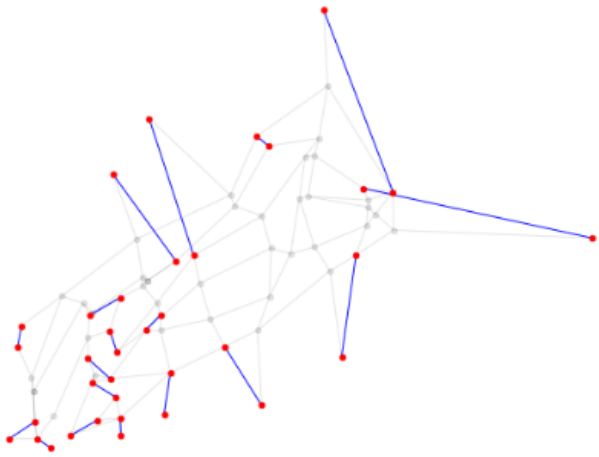

Figure 5: Min Weight Matching in Original Graph.

3.3: Compute Eulerian Circuit

Now that much of the preprocess work is done due to the odd degree nodes. We can safely now use NetworkX package of the Eulerian circuit to find a circuit of nodes making the least possible route covering all the nodes. However, there are still some limitations to be fixed when we compute the eulerian route because the augmented graph likely contains some edges that do not exist on the original graph.

Hence we first will simply apply the Eulerian circuit function on our data. The resulting answer is just a list of tuples which represent node pairs. We then loop through each edge in the Eulerian circuit. Wherever there is an edge that does not exist in the original graph, we replace it with the sequence of edges comprising the shortest path between its nodes using the original graph. This way our correct eulerian circuit is computed.

Fig 6 shows the final hiking trail map solution marked with numbered edges to move from beginning to end (from 0 to 157 directions). Multiple numbers indicate trails we must double back on. In order to view the above solution more clearly, a visualization movie can be created. First PNG images can be produced for each direction (edge walked) from the solution. Then the PNG images can be stitched together to make the gif of the directions of each move. This will give a better visualization of the directions to move next from node to node.

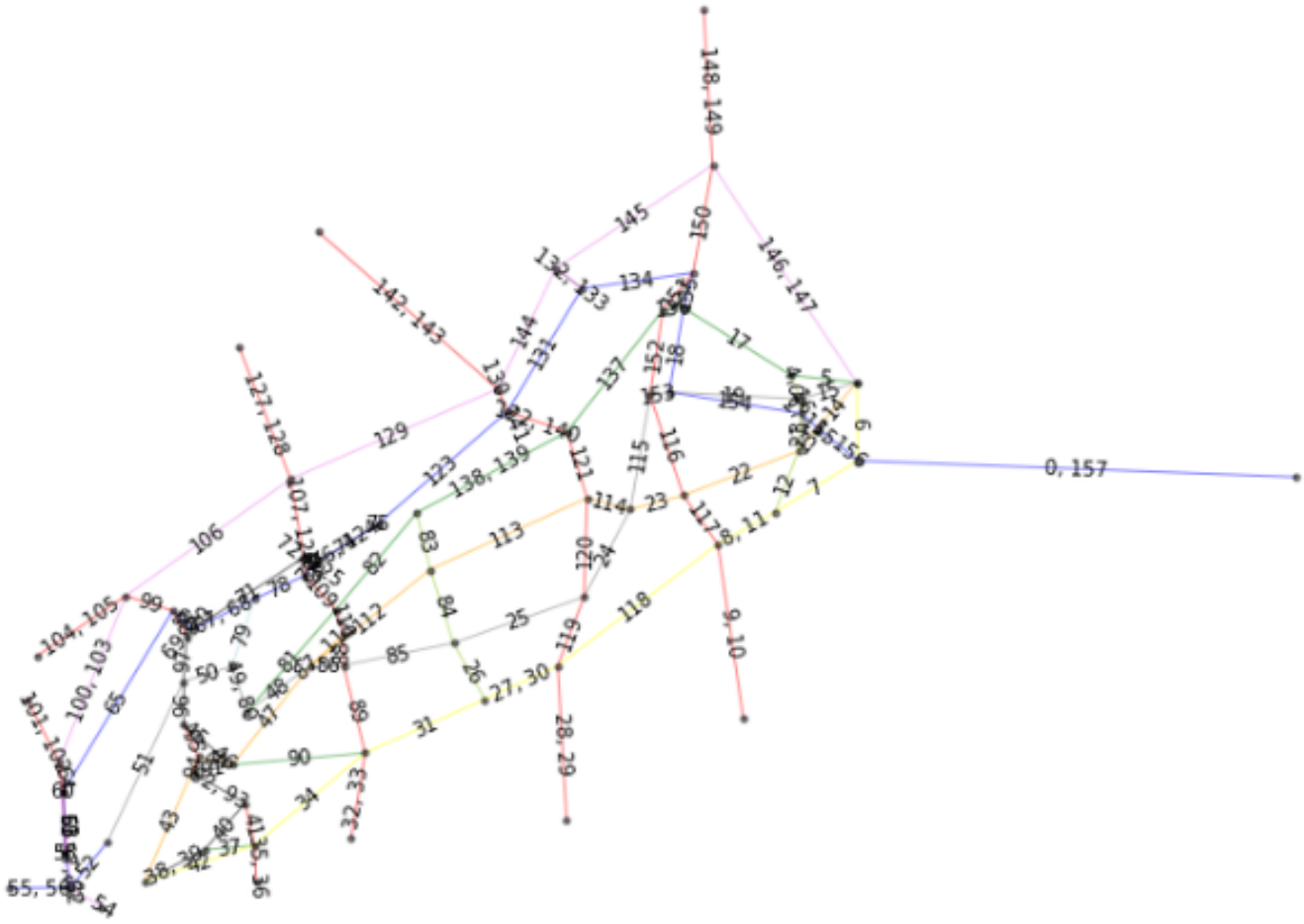

Figure 6: Trail map with sequence of edges indicated.

4 Conclusion

In this report, a network of hiking trail map in the town of Hamden, the United States famously known as the Sleeping Giant Park has been analyzed to depth providing an optimized strategy for hikers of the park to move along the directions such that each trail of the park is visited with least double backing.

Starting from the raw data of edges and nodes, the graph was plotted to view the network structure of the trails in the park. Doing that, degree measure of the nodes were computed and odd degree node pairings were formed. A total of 36 nodes were odd out from 123 nodes. A pair combination of the 36 odd nodes came out to be 630 which in fact are the edges connecting the odd nodes to every other odd node. Dijkstra's theorem was applied to find the minimum distances of all these 630 edges and then weight minimum function was applied to only catch the least weighted edge of every odd node. Doing that, the graph was then augmented on the real graph. The Eulerian circuit was computed and then recomputed over the disillusioned circuit replacing all the edges that did not exist in the real graph with those that existed.Hence finally, our final solution of a traveling salesman problem was achieved in the form of a numbered route map sequencing the trails to move from starting position (0th) till the end (157th) making a complete circuit in the network.

All of the above steps of the travelling salesman problem performed in this report makes an easy methodology to follow for any problem of optimizing route strategy given the relevant data. This report thus provides a baseline, whereby, more such networks could be analyzed and explored for best-optimized travelling strategy.[3][4]

References

[1] P. Orponen and H. Mannila, "On approximation preserving reductions: complete problems and robust measures," 1987.

[2] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.

[3] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, vol. 4, no. 3, pp. 237–244, 1977.

[4] C. H. Papadimitriou and M. Yannakakis, "The traveling salesman problem with distances one and two," *Mathematics of Operations Research*, vol. 18, no. 1, pp. 1–11, 1993.