Constructing a Traffic Network from Scratch

Shengfeng Xue

Abstract

The shortest path tree, obtained by Dijkstra’s algorithm, is often used in the analysis of traffic networks to get an overview of how the traffic flows in the graph. Beyond the analysis, it can also be used to construct a network from scratch. Though there are no edges in the beginning, initial base edges can be added using heuristics. Also, to construct a complete graph, based on the base edges, more edges, as well as adjustments, will be done to optimize the graph in terms of construction cost (sum of lengths of all edges) and detour cost (sum of how longer traffic flows without a direct edge than in the case with one). The goal of the project is to construct a traffic network from scratch while striking a balance between the construction cost and the detour cost.

1. Preface

In fields like public transportation, especially aviation, the traffic network is the basis for all other things. To build a traffic network, multiple factors should be considered, including the cost of construction, the profit of an edge, the minimal requirements for traffic flow, etc. In building a network from scratch, every step should be fully considered so that the sum of all costs can be minimized.

1. Modelling
2. Nodes

The nodes represent cities (In this case US cities) by a tuple of latitude and longitude.

1. Edges

Edges represent links (flights) between cities in which traffic flows go in their shortest paths for their destinations.

1. Traffic

The traffic data is kept in a separate data structure (not in the nodes) that can be either a list or a matrix. In the case of a list, it is represented as (city1->city2:traffic). In the case of a matrix, it is represented as “table[city1][city2]=traffic”. Initially, when traffic data is read from a .csv file, it is in the form of a list. But in the next steps, it will change in form to a matrix for convenience in calculation.

1. Goal

Given a set of nodes (latitude,longitude,name), traffic (in either of the 2 forms), connect all edges (the length of an edge is decided by the geographical coordinates of 2 nodes) so that the construction cost and the detour cost are balanced.

1. Methods

**Adding the basic necessary edges**

To start from scratch, it is necessary to decide the edges to start with. In real life, the best flights to start with are those that carry a large number of passengers with a relatively low cost, for which they are labeled “strong links”. In this case, the strength of a link is calculated as the quotient of the number of direct passengers (not considering those who transferred or are to transfer). Since the links that send the most passengers from their origins to their destinations are both profitable and necessary (The passengers do not want to stop at another city for such necessary journeys), they lay the foundation for the whole graph, and are to be added to the graph first. What’s more, the process will return not only the base graph, but also the “unsolved” traffic for which direct link is yet to be set.

**Analyzing unsolved traffic**

Before the analysis, a calculation is to be done to get the cost of detour of unsolved traffic. Detour is defined as the product of traffic amount and the difference between the actual travelling distance and the straight-line distance. If the cost is small, the traffic will not be the first to be addressed as it implies that the journey only took a small detour and setting a direct link will not be worthy. If it is too large, it will not be put in the first place either as setting a link is too costly, and cost-efficient approaches lies in setting other shorter links.

To find the first to be addressed, set a threshold (in this case 10000) and find the traffic from city a to b that has the smallest detour cost above the threshold. Then, find a solution to it by adding new links (for details, see the next section). After the link is set, the 2nd traffic to be addressed will first assess its new detour cost. If the cost falls below the threshold, it’ll be dismissed as “solved”. Otherwise, a solution will be found like what was done in the previous case. Loop until all traffic cases are solved. Picking the cost that is the smallest above the threshold is algorithmically safe, as all traffic costs decrease or stay the same when a new link is added.

**Finding a solution for unsolved traffic**

This is the key factor in the building of the graph, as it seeks to strike a balance between construction costs and traffic (detour) costs. The overall cost for the route is the sum of 2 parts: Construction cost, which is in proportion to the length of links to be added. Travel cost, which is the product of the amount of traffic and the overall travelling length. The core lies in finding a transferring node that minimizes the overall cost. The process is done in 2 directions: For name1 and name2 to be solved, first do: Adding a new link starting from name1 to a transfer node that is connected to name2, then, do this starting from name2 to name1. The process will finally return as a plan of 3 node names and a cost, which is to be used as the criterion for the sorting of plans.

**Further extension of the graph**

After the initial graph is all set, further extensions can be done to the graph, which shows the elasticity of the graph. Similar to previous steps, adding a new node and new traffic generated by it will induce “finding a solution”, but it is done in only 1 direction from the new node to other nodes, as the method requires the destination node to be already in the graph. What’s more, this can be done with multiple destinations, which means that for every transfer node, the overall cost is added up, and the node with the least sum of costs is chosen.

1. Results

Dataset:

Nodes: <https://simplemaps.com/data/us-cities>

Traffic: Defined manually to test the cases, as using regular real-world datasets will result in strongly-connected cities, which is against the goal of this project that tries to minimize the number of links to achieve lower costs. This can be described as “when only a few people are flying, and planes are limited in capacity and travel distance.”

About the dataset: The dataset is divided into training dataset and testing dataset. In this case, in the first step, it is only the training dataset that is used to build the base network. After this, the testing dataset is used to add new nodes and edges to the network.

Hyperparameter tuning is used. The 3 hyperparameters are: The coefficient coef1 for basic edges, the coefficient coef2 related to edge construction cost for additional edges, and the threshold of detour cost for solving detours. The larger coef1 is, the more basic edges are added. The larger coef2 is, the more discouraged newly-added nodes are to be directly connected to their destinations. For the threshold, if it is too low, excessive number of edges will be added just to unilaterally reduce detour cost but at the cost of construction cost. If it is too high, unnecessarily long direct links with inadequate traffic, while leaving short gaps unfilled.

For the tuning below, when the total cost is used, coef2 is fixed to 15, as it decides how the total cost is counted, making cases with this hyperparameter different non-comparable. It should be clear that coef2 can still be tuned, as the fixed coef2 is only for calculating the total cost, not governing whether a direct link should be constructed.

Case 1: coef1: 15, coef2: 15, threshold: 10000

Chart, line chart

Description automatically generatedChart

Description automatically generated

Fig 3-1 Basic edge graph and complete graph of case 1. The 2 axes represent the longitude and the latitude. It is obvious that the basic edge graph is far from complete, with important edges missing. The complete graph makes good sense, even though it has some long edges.

Sum of lengths of all edges: 58853. Detour cost: 37158. Total cost: 919953

Chart

Description automatically generated

Fig 3-1-1 Adding a new node to the graph and setting the optimal edge (only 1) to minimize overall costs.

Case 2: coef1: 15, coef2: 15, threshold: 20000

Chart

Description automatically generated

Fig 3.2 Complete graph in case 2. Less edges lead to more detour cost.

Sum of lengths of all edges: 54519. Detour cost: 64206. Total cost: 881991

Case 3: coef1: 15, coef2: 15, threshold: 30000

Graphical user interface, text

Description automatically generated

Fig 3.3 Complete graph in case 3.

Sum of lengths of all edges: 50185. Detour cost: 137454. Total cost: 890229

With the first 2 hyperparameters fixed, as the cost threshold increases, the total cost drops then increases, meaning that an optimal value for the threshold can be found by trials, which is, in this case, 20000.

Case 4: coef1: 15, coef2: 30, threshold: 20000

Graphical user interface

Description automatically generated

Fig 3.4 Complete graph in case 4.

Sum of lengths of all edges: 51800. Detour cost: 130947. Total cost: 907947

The coef2 should not be large, or it will induce heavy detour cost while not saving too much edge length. In this tuning, values like 12,10,7.5,5,2 are tried, but they all yield a result the same as that in case 2, meaning that with cost threshold fixed at a large value, if coef2 is small enough, it will yield an optimum.

Case 5: coef1: 15, coef2: 7.5, threshold: 10000

Graphical user interface

Description automatically generated

Fig 3.5 Complete graph in case 5.

Sum of lengths of all edges: 59706. Detour cost: 29738. Total cost: 925328

Compared to case 1, the total cost rises as a result of excessive edges. Coef2, like the threshold, have an optimal value with other coefficients fixed. However, sometimes, a change in coef2 may not change the complete graph.

Case 6: coef1: 20, coef2: 7.5, threshold: 20000

Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated

Fig 3-6 Basic edge graph and complete graph of case 6.

Sum of lengths of all edges: 58389. Detour cost: 87827. Total cost: 963662

Lowering the threshold for the base edges only adds to both costs.

WARNING: The coef1 should not be too low, or the base edges will not form one connected graph.

Case 7: coef1: 13.5, coef2: 15, threshold: 10000

A screenshot of a computer

Description automatically generated Graphical user interface, text

Description automatically generated

Fig 3-7 Basic edge graph and complete graph of case 7.

Sum of lengths of all edges: 56679. Detour cost: 54648. Total cost: 904833

Coef1 is a dangerous coefficient. In theory, the lower it is, the lower the total cost may be. However, for a certain set of nodes and traffic, it has a limit beyond which the base edges will not form one connected graph for later steps to be done.

**Analysis and Regression**

In addition to the cases above, several other cases are also tested. Below is a plot of the 2 costs which are chosen because neither of them is affected by any parameter, and they both represent the overall evaluation of a graph.

Chart 3: Sum of Lengths of Edges VS Detour Cost

If the total cost is calculated by a linear combination of the 2 costs: total\_cost=k\*sum\_of\_lengths \_of\_all\_edges+detour\_cost, case 2 will be the optimal case if k (construction cost per km) lies within (6.65,16.90).

Since the network is constructed using linear means (basic edges, additional edges and adjustments, as well as the 2 metrics), the graphs yielded by hyperparameter tuning have the 2 metrics in a certain range and suitable for linear regression. The points in the plot only represent a small close-to-straight-line segment of a long curve that takes too long unnecessary steps and time to get. What this project has done is just to find a set of candidates.

**Issues beyond calculation**

An issue in this project is that it lacks removing (selling) an edge for lower construction cost. However, this does not stand in the real world as strong links must be preserved, for people don’t want to stop at any other node for a journey that takes many people directly to their destinations. However, in the real world, the calculation for the cost of transferring and waiting is beyond this project’s calculations, as the waiting time in the real world varies in a large range from no more than 1 hour to more than 3 hours, and accidents may add up to the hours. Considering the need of the people in the real world, edges set in the first step, which are called “base edges”, should not be removed as a moral requirement. The complete graph is built on condition that some strong edges must exist. It should also be noted that the graph is heavily dependent on the traffic data, as any change in the traffic data can affect the graph in essential ways.

1. Conclusion

By doing this project, these are what I’ve learned.

1. Plan, explore, and yield results in advance. Do not leave things too late.
2. Heuristic methods can help reduce time costs and approach to the candidate cases.
3. For algorithms executed on a large graph, it is necessary to devise ways to minimize time cost by avoiding unnecessary operations while keeping the outcomes correct by maintaining strict logic.
4. Not only should something be constructed, but also it should be evaluated. This is logical integrity.
5. Hyperparameter tuning adds to the interest of the project.
6. For a project to be connected with the real world, moral issues should be considered.

References

[1] Paul, Abraham & Venkataraman, Saravanan & Jeba Thangaiah, Ranjit. (2011). Data Mining Analytics to Minimize Logistics Cost. International Journal of Advances in Science and Technology. 2. 89.

[2] Amin Hashemi, Mohammad Bagher Dowlatshahi, Hossein Nezamabadi-pour, MGFS: A multilabel graph-based feature selection algorithm via PageRank centrality, Expert Systems with Applications, Volume 142, 2020, 113024, ISSN 0957-4174.

[3] Sciarra, C., Chiarotti, G., Laio, F. et al. A change of perspective in network centrality. Sci Rep 8, 15269 (2018).

[4] Abhijit Bhattacharya, Anurag Kumar, A shortest-paths-tree based algorithm for relay placement in a wireless sensor network and its performance analysis, Computer Networks, Volume 71, 2014, Pages 48- 62, ISSN 1389-1286

[5] P. Narvaez, Kai-Yeung Siu and Hong-Yi Tzeng, "New dynamic algorithms for shortest path tree computation," in IEEE/ACM Transactions on Networking, vol. 8, no. 6, pp. 734-746, Dec. 2000, doi: 10.1109/90.893870.

[6] W. Bechkit, M. Koudil, Y. Challal, A. Bouabdallah, B. Souici and K. Benatchba, "A new weighted shortest path tree for convergecast traffic routing in WSN," 2012 IEEE Symposium on Computers and Communications (ISCC), 2012, pp. 000187-000192, doi: 10.1109/ISCC.2012.6249291.

[7] Zhang, Baoxian and Mouftah, Hussein T. ‘Destination-driven Shortest Path Tree Algorithms’. 1 Jan. 2006 : 123 – 130. Print.