Network Algorithms final project

Atifa Agazada , Amina Hajieva , Nafila Amirli

French Azerbaijani University

2021

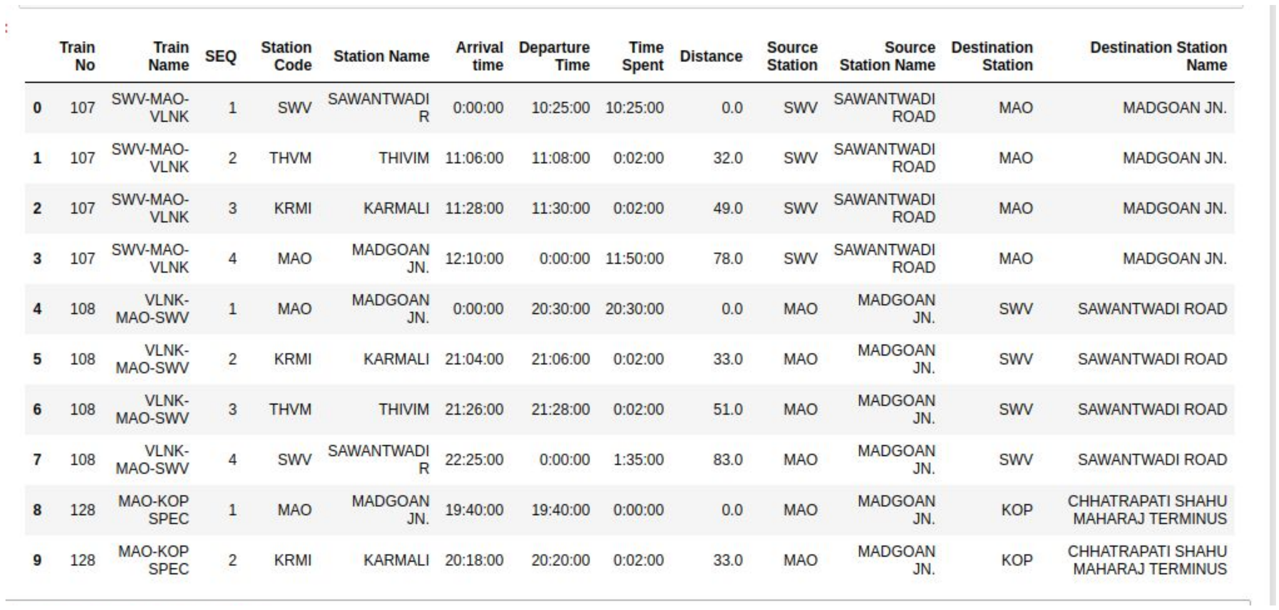
Table of Contents

Introduction

The purpose of this project is analyzing the train stations of India in Python using graph theory and including using different algorithms such as BFS DFS Dijkstra Bellman-Ford and etc. In this report the used algorithms, ways are discussed and shown some results. On the other hand, the code has been commented for better understanding.

Data Description

Since the transport methods are good fit to analyze as a Graph, we have used train data where we have columns like this:



	Train No	Train Name	SEQ	Station Code	Station Name	Arrival time	Departure Time	Time Spent	Distance	Source Station	Source Station Name	Destination Station	Destination Station Name
0	107	SWV-MAO-VLNK	1	SWV	SAWANTWADI R	0:00:00	10:25:00	10:25:00	0.0	SWV	SAWANTWADI ROAD	MAO	MADGOAN JN.
1	107	SWV-MAO-VLNK	2	THVM	THIVIM	11:06:00	11:08:00	0:02:00	32.0	SWV	SAWANTWADI ROAD	MAO	MADGOAN JN.
2	107	SWV-MAO-VLNK	3	KRMI	KARMALI	11:28:00	11:30:00	0:02:00	49.0	SWV	SAWANTWADI ROAD	MAO	MADGOAN JN.
3	107	SWV-MAO-VLNK	4	MAO	MADGOAN JN.	12:10:00	0:00:00	11:50:00	78.0	SWV	SAWANTWADI ROAD	MAO	MADGOAN JN.
4	108	VLNK-MAO-SWV	1	MAO	MADGOAN JN.	0:00:00	20:30:00	20:30:00	0.0	MAO	MADGOAN JN.	SWV	SAWANTWADI ROAD
5	108	VLNK-MAO-SWV	2	KRMI	KARMALI	21:04:00	21:06:00	0:02:00	33.0	MAO	MADGOAN JN.	SWV	SAWANTWADI ROAD
6	108	VLNK-MAO-SWV	3	THVM	THIVIM	21:26:00	21:28:00	0:02:00	51.0	MAO	MADGOAN JN.	SWV	SAWANTWADI ROAD
7	108	VLNK-MAO-SWV	4	SWV	SAWANTWADI R	22:25:00	0:00:00	1:35:00	83.0	MAO	MADGOAN JN.	SWV	SAWANTWADI ROAD
8	128	MAO-KOP SPEC	1	MAO	MADGOAN JN.	19:40:00	19:40:00	0:00:00	0.0	MAO	MADGOAN JN.	KOP	CHHATRAPATI SHAHU MAHARAJ TERMINUS
9	128	MAO-KOP SPEC	2	KRMI	KARMALI	20:18:00	20:20:00	0:02:00	33.0	MAO	MADGOAN JN.	KOP	CHHATRAPATI SHAHU MAHARAJ TERMINUS

Data has been obtained from open resources of Indian governments which describes the train stations and their connections.
https://data.gov.in/resources/indian-railways-time-table-trains-available-reservation-03082015
Our initial data was dirty: for example inside Distance columns there were non numerical and NA values so that we cleaned it in order not to have further problems while calculations and etc.
In general, there are 186116 rows × 13 columns.

Used algorithms for solving the problems

Degree Connectivity

Degree Connectivity is the number of edges connected to a node. In the case of a directed graph, we can have 2-degree centrality measures: **Inflow Centrality** and **Outflow Centrality**

Closeness Centrality

In a connected graph,closeness centrality (or closeness) of a node is a measure of centrality in a network, calculated as the sum of the length of the shortest paths between the node and all other nodes in the graph. Thus the more central a node is, the closer it is to all other nodes.

Network Density

Network Density is based on how many edges a graph has; we can use the formula for directed graphs:

$$\frac{\#\text{edges}}{N \cdot (N-1)} = \frac{1}{N \cdot (N-1)} \cdot \sum_{n=1}^{N} \mathbf{deg}(\text{node}_n)$$

Breadth-first search algorithm

Breadth-first Algorithm:

- Depth-first: visit all neighbors before visiting neighbors of your neighbors
- Start from node *s*.
- Visit all neighbors of node *s*.
- Then visit all of their neighbors, if not already visited
- Continue until all nodes visited

Intuition of Implementation

- Keep a queue of nodes to visit
- When visiting node *u*, put neighbors of *u* on the queue, if neighbor not yet visited
- Queue contains all nodes that have been seen, but not yet visited

Depth-first search algorithm

Another common type of graph algorithm is a **depth-first** algorithm

- Depth-first: visit all neighbors of a neighbor before visiting your other neighbors
- First visit all nodes reachable from node *s* (ie visit neighbors of *s* and their neighbors)

- Then visit all (unvisited) nodes that are neighbors of *s*
- Repeat until all nodes are visited

Algorithm:

- Don't use a queue
- Instead, mark nodes as to their status
    - **White**: not yet seen (initial color)
    - **Gray**: seen, but not finished
    - Node *u* is **finished** if we have visited all nodes reachable from *u*
    - **Black**: finished

For some algs, the nodes also have start and finish stamps

- These stamps give the time (ie step in the algorithm) when each node is first seen and when it is finished

Bellman-Ford algorithm

The algorithm has a running time of O(mn) where n is the number of nodes and m is the number of edges. It is slower than Dijkstra but can handle negative edge weights.

Parameters:

- G (NetworkX graph) – The algorithm works for all types of graphs, including directed graphs and multigraphs.
- source (node label) – Starting node for path
- weight (string, optional (default='weight')) – Edge data key corresponding to the edge weight

Returns:

- pred, dist – Returns two dictionaries keyed by node to predecessor in the path and to the distance from the source respectively.

Djikstra's algorithm

Here is how Dijkstra's algorithm works:

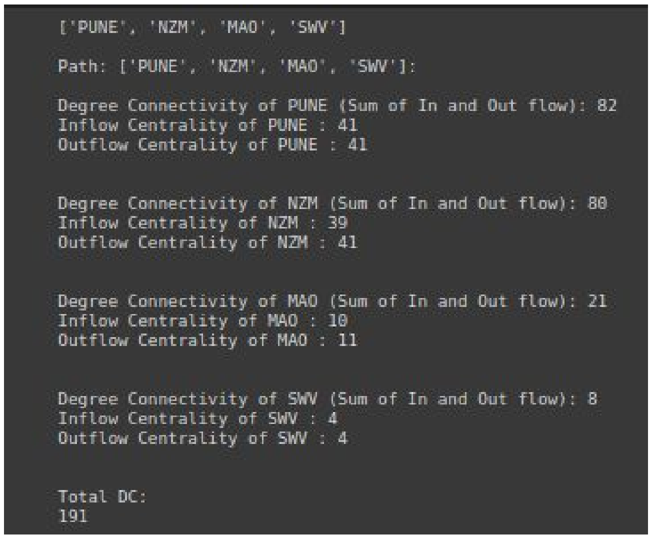
1. Mark all nodes unvisited and store them.
2. Set the distance to zero for our initial node and to infinity for other nodes.
3. Select the unvisited node with the smallest distance, it's current node now.
4. Find unvisited neighbors for the current node and calculate their distances through the current node. Compare the newly calculated distance to the assigned and save the smaller one. For example, if the node A has a distance of 6, and the A-B edge has length 2, then

the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8.
5. Mark the current node as visited and remove it from the unvisited set.
6. Stop, if the destination node has been visited (when planning a route between two specific nodes) or if the smallest distance among the unvisited nodes is infinity. If not, repeat steps 3-6.

Results

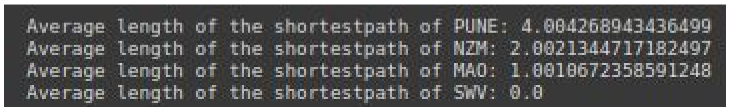
Results due to metrics and algorithms

1. **Degree Connectivity**



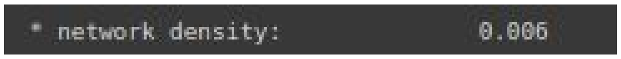
```
['PUNE', 'NZM', 'MAO', 'SWV']

Path: ['PUNE', 'NZM', 'MAO', 'SWV']:

Degree Connectivity of PUNE (Sum of In and Out flow): 82
Inflow Centrality of PUNE : 41
Outflow Centrality of PUNE : 41


Degree Connectivity of NZM (Sum of In and Out flow): 80
Inflow Centrality of NZM : 39
Outflow Centrality of NZM : 41


Degree Connectivity of MAO (Sum of In and Out flow): 21
Inflow Centrality of MAO : 10
Outflow Centrality of MAO : 11


Degree Connectivity of SWV (Sum of In and Out flow): 8
Inflow Centrality of SWV : 4
Outflow Centrality of SWV : 4


Total DC:
191
```

2. **Closeness Centrality**



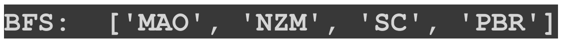
```
Average length of the shortestpath of PUNE: 4.004268943436499
Average length of the shortestpath of NZM: 2.0021344717182497
Average length of the shortestpath of MAO: 1.0010672358591248
Average length of the shortestpath of SWV: 0.0
```

3. **Network Density**



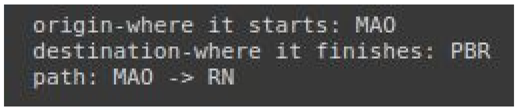
```
* network density:              0.006
```

4. **Breadth-first search algorithm**



```
BFS:  ['MAO', 'NZM', 'SC', 'PBR']
```

5. **Depth-first search algorithm**

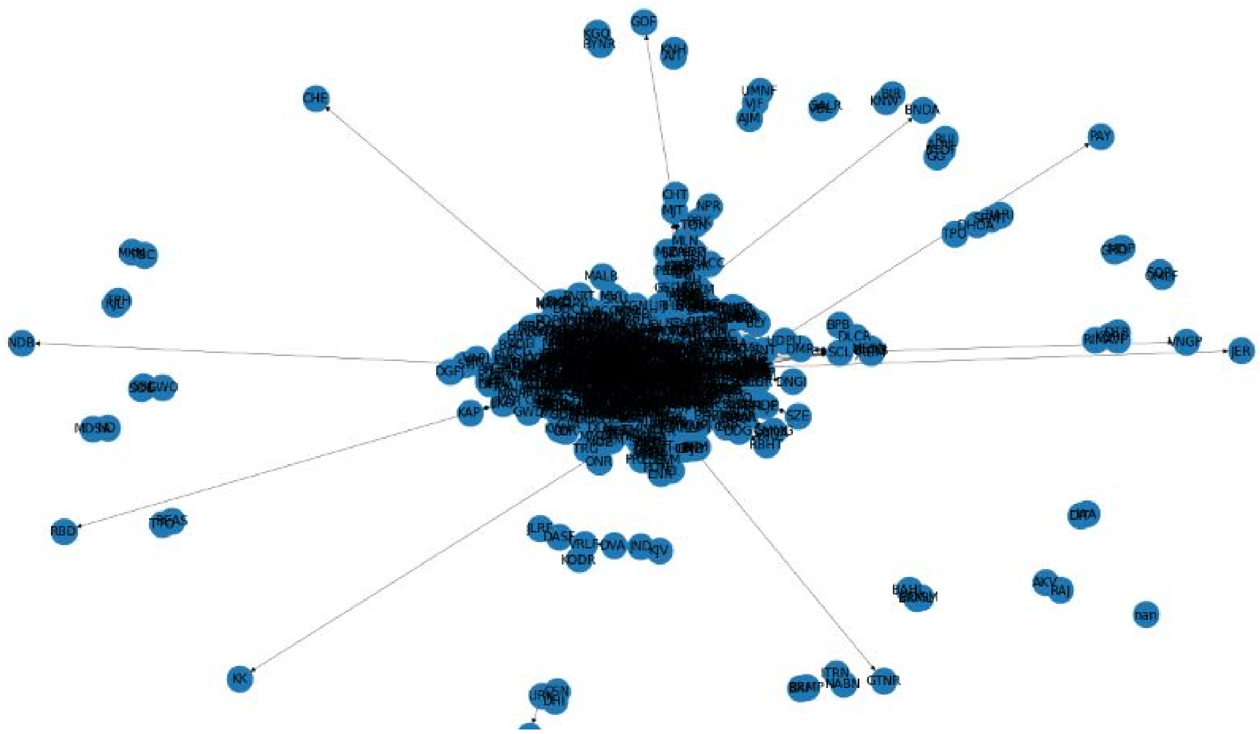


```
origin-where it starts: MAO
destination-where it finishes: PBR
path: MAO -> RN
```

Data visualization

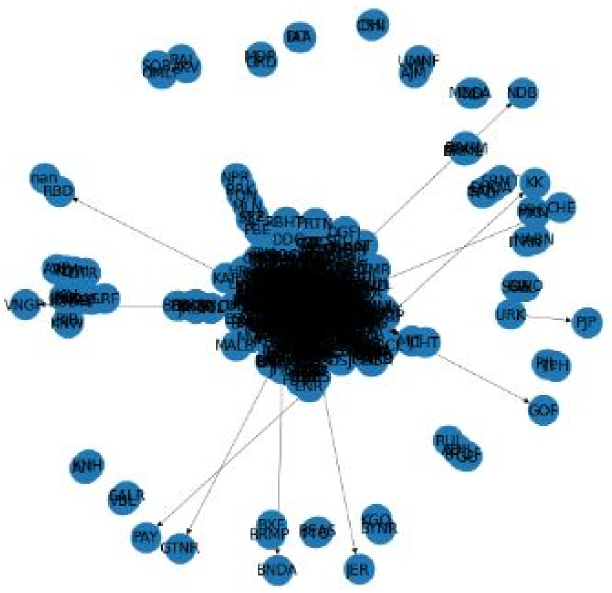
Here are the plots that appears as an output while running the program:

1. **Graph representation of the train paths.**



2. **Shortest path for 2 destination: PUNE and SWV**

```
Shortest Path: ['PUNE', 'NZM', 'MAO', 'SWV']
```

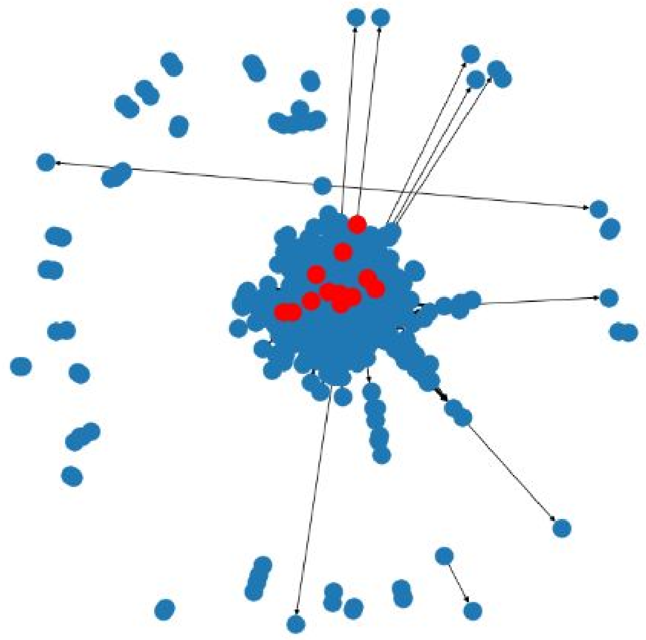


3. **Shortestpath due to Distance:** `['PUNE', 'KJT', 'KYN', 'DR', 'SWV']`

So, When we started to cosider the Distance attribute, the shortest path changed as a result.

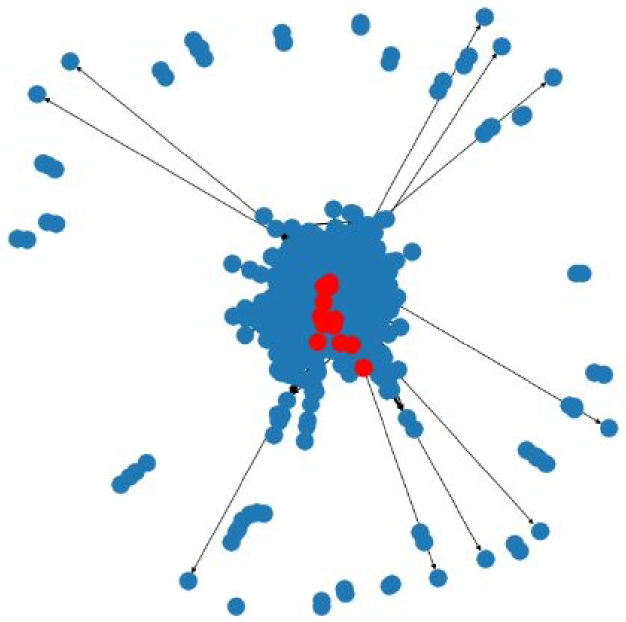
4. **Shortest Path by Bellman-Ford from MAO to PBR**

```
1889.0 ['MAO', 'SWV', 'DIVA', 'PNVL', 'DRD', 'VR', 'ST', 'BRC', 'ANND', 'ADI', 'SMNH', 'PBR']
```



5. **Shortest Path by Dijkstra from MAO to PBR**

```
1889.0 ['MAO', 'SWV', 'DIVA', 'PNVL', 'DRD', 'VR', 'ST', 'BRC', 'ANND', 'ADI', 'SMNH', 'PBR']
```



Conclusion

In this project varios graph algorithms have been used to find the shortest path between two stations. By using Breadth-first search and Depth-first search algorithms, we have concluded that BFS is less optimal than DFS.

Moreover, after using both Dijkstra and Bellman-Ford Both algorithms, we can conclude that They can solve the problem of finding the quickest path between two nodes however Bellman-Ford just can be eliminated since there is absence of negative edge weights on the graph,thus Dijkstra is an efficient way to plan routes in train networks for realistic conditions and a transportation optimization method based on Dijkstra's algorithm is used to find the optimal minimal time path and to simplify cost calculations.

Google Colab Link

https://colab.research.google.com/drive/17kM0eTXI6t--Dx9Jyi-C61y_ZxXdBjaZ?usp=sharing#scrollTo=5Vtk1rpCQJRG