Yanaal Niyazi
Professor Leonidas Konthothanassis
DS210
May 2024

# Pennsylvania Road Network Analysis - Project Report

Link to dataset: https://snap.stanford.edu/data/roadNet-PA.html

How to run program:

1) Download "main.rs" and "graph.rs" and place it into a directory that you will use.
2) Download the dataset "roadNet-PA.txt.gz" from the link above and save it in the same directory as "main.rs" and "graph.rs".
3) CD to the directory and type in "cargo build".
4) Finally, proceed to run the command "cargo run --release".

## *Introduction*

This project aims to analyze the road network data of Pennsylvania using graph theory techniques implemented in Rust. The dataset consists of over 1 million nodes and edges, where nodes represent the intersections or endpoints of roads, and edges represent connections between these intersections, providing valuable insights into the structure and characteristics of the road network. I found this specific dataset to be intriguing as I thought about the value it would provide to businesses located there and the government as a result of analyzing this data thoroughly, which I will discuss later on in this report.

## *Overview*

The project consists of Rust code that reads the road network data from a file, constructs an undirected graph, and performs various analyses on the graph, which includes calculating degree centrality, degree of nodes, and average distance. These analyses provide insights into the importance of nodes in the network, connectivity, and average distances between nodes.

## *Modules and functions*

1. main.rs:

main: Entry point of the program. Reads the graph from the file, calculates degree centrality, degree of nodes, and average distance, and displays the top nodes by degree centrality and degree.

top_nodes_centrality: Displays the top nodes by a given centrality measure (degree centrality in this case).

top_nodes_degrees: Displays the top nodes by degree.

2. graph.rs:

read_lines: Reads lines from a file.

read_graph_from_file: Reads the road network data from a file and constructs an undirected graph.

calculate_degree: Calculates the degree of each node in the graph.

calculate_degree_centrality: Calculates the degree centrality of each node in the graph.

calculate_average_distance: Calculates the average distance between random pairs of nodes in the graph.

bfs_distances: Performs breadth-first search to find the distances from a given source node to all other nodes in the graph.

*Analysis*

```
Top 10 Nodes by Degree Centrality:
1. Node 93600: 0.000002157297
2. Node 3491381: 0.000002157297
3. Node 127088: 0.000002157297
4. Node 1165235: 0.000002157297
5. Node 2313426: 0.000001917597
6. Node 3841582: 0.000001917597
7. Node 2163082: 0.000001917597
8. Node 165264: 0.000001917597
9. Node 41680: 0.000001917597
10. Node 943227: 0.000001917597
```

These are the top 10 nodes with the highest 'degree centrality', indicating they are crucial hubs in the Pennsylvania Road Network. Businesses and the government can leverage this information for various purposes:

- Tourism Hotspots: Nodes with high degree centrality are likely to be major tourist attractions or transportation hubs. Tourist agencies can focus on promoting these areas to attract more visitors.

- Road Efficiency: Government agencies can prioritize road maintenance and expansion around these hubs to improve overall road efficiency and reduce traffic congestion.

- Advertisement Placement: Businesses can strategically place advertisements on billboards near these high-traffic nodes to maximize visibility and reach.

```
Top 10 Nodes by Degree:
1. Node 127088: 9.00
2. Node 3491381: 9.00
3. Node 93600: 9.00
4. Node 1165235: 9.00
5. Node 2163082: 8.00
6. Node 1000477: 8.00
7. Node 2313426: 8.00
8. Node 2497251: 8.00
9. Node 3052243: 8.00
10. Node 3261478: 8.00
```

These are the top 10 nodes have the highest 'degrees', indicating they have the most connections in the network. This information can be used in similar ways as degree centrality for tourism, road efficiency, and advertisement placement.

```
Average Distance: 0.43
```

The average distance between random pairs of nodes in the Pennsylvania Road Network is approximately '0.43'.

To be able to calculate this value, I employed the Breadth-First Search Algorithm. In general, '0.43' indicates a relatively short distance between nodes and this suggests that the road network is well-connected allowing for efficient travel.

- Efficient Routing: Businesses can use this information of short distances to optimize delivery routes or transportation services, minimizing travel time and costs.

- Infrastructure Planning: Government agencies can identify areas with large average distances and prioritize building new roads or improving existing ones to enhance connectivity.

- Location Planning: Businesses can choose strategic locations for new branches or facilities based on proximity to nodes with high degrees or centrality, ensuring easy access for customers and suppliers.

*Conclusion*

In conclusion, I analyzed a dataset of road networks using Rust and calculated the degree and centrality of the top 10 nodes from the random sample of the dataset each time the program is run. Through degree centrality and node degrees, we identified crucial hubs and highly connected nodes, offering valuable information for tourism, road maintenance prioritization, and advertisement strategies. Moreover, the calculated average distance of between nodes indicates a well-connected network, facilitating efficient routing, infrastructure planning, and strategic business location decisions. This project highlights the practical applications of graph theory in optimizing transportation networks for businesses and governmental agencies.

*Sources used*


https://towardsdatascience.com/notes-on-graph-theory-centrality-measurements-e37d2e49550a

https://omicsforum.ca/t/what-are-the-differences-between-degree-and-betweenness-centrality/163

https://docs.rs/graphrs/latest/graphrs/

https://docs.rs/petgraph/latest/petgraph/

https://doc.rust-lang.org/std/collections/struct.VecDeque.html

https://doc.rust-lang.org/std/collections/struct.HashMap.html