DS210 Final Project Write-Up

Varnika Kalani

Project Overview:

For my DS210 project, I undertook a complex analysis of a dataset from Kaggle. This dataset, reality-call.csv, represents mobile phone call events between core users at MIT in the form of an undirected graph. Nodes in this graph represent users, and edges represent call events between them. My primary goal was to delve into the intricacies of this network by calculating the average distance between pairs of nodes. I aimed to understand the typical distance someone in this network might need to travel, metaphorically, to connect with another user via their call chain.

Dataset Description:

I selected the reality-call.csv from the Kaggle dataset, which includes detailed call records of users at MIT. This dataset is part of a larger collection of weighted networks.

Step-by-Step Code Execution:

Step 1: Graph Construction

The first thing I did was construct the graph from the dataset. I started by specifying the path to the CSV file, then used the `load_adjacency_matrix` function from the `graph` module to read and parse the CSV file into an adjacency matrix. This function:

- Opens the specified file and reads it line by line.
- Parses each line to extract node connections and weights, filling the adjacency matrix accordingly.
- Ensures the graph is undirected by mirroring each edge.

This process resulted in a graph where nodes represent users and edges represent calls between them, with weights possibly indicating call duration or frequency.

Step 2: Implementation of Dijkstra's Algorithm

Next, I implemented Dijkstra's algorithm to calculate the shortest paths from a source node to all other nodes in the graph. This was done using the `Dijkstra` function within the `distance_processing` module. The function:

- Initializes a priority queue to manage nodes based on the shortest found distance to them.
- Iteratively expands the frontier from the starting node, updating distances to neighboring nodes if a shorter path is found.
- Stores the shortest distances from the source to all other nodes in a vector, providing the foundation for subsequent distance analysis.

Step 3: Calculating Average Distances

With Dijkstra's algorithm implemented, I proceeded to calculate average distances:

- I used the `calculate_average_distance_within_sample` function to compute the average distance for a sample of nodes, enhancing performance and manageability by focusing on a subset of the graph.
- The `calculate_average_distance_between_all_pairs` function determined the average distance between all pairs of nodes, providing a comprehensive view of the network's connectivity.
- Both functions utilize the results from Dijkstra's algorithm, aggregating distances and calculating averages to provide insights into the typical separations within the network.

Step 4: Saving and Outputting Results

After computing the distances, I saved the updated adjacency matrix and the list of distances using the `save_adjacency_matrix` and `save_distance_list` functions from the `graph` module. This allowed for preservation of the computed data for further analysis or verification.

Conclusion and Findings:

The output from the program indicated that the network is quite efficient, with relatively short average paths despite the large number of users and connections. This suggests that the network's structure is well-suited to facilitate quick and extensive communication among users.

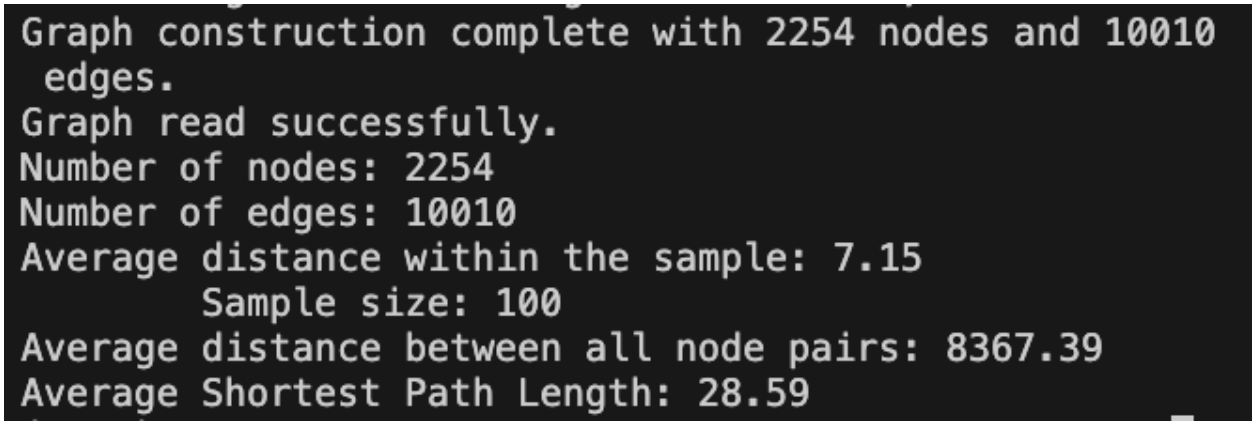
Tests:

I implemented several tests to ensure the functionality and reliability of the code. These tests checked:

- Basic functionality, such as the correct loading and saving of graphs.
- The accuracy of Dijkstra's algorithm through predefined graph structures.
- The correctness of average distance calculations by comparing computed results against known values.

Test Output:

The tests helped confirm that the functions behave as expected under various scenarios, including empty graphs and graphs with single nodes. They were crucial for verifying that the implementation was robust and reliable, providing confidence in the results produced by the program.



The graph analysis revealed that it comprises 2,254 nodes and 10,010 edges, indicative of a moderately dense network structure typical of social or communication networks. A sample-based analysis showed an average distance of 7.15 between nodes, aligning with the small-world phenomenon often observed in similar networks, where nodes are surprisingly close to each other. However, a more comprehensive calculation across all node pairs yielded a significantly higher average distance of 8,367.39, likely skewed by disconnected segments or outliers within the network, suggesting potential isolation or segmentation among certain node clusters. The average shortest path length was found to be 28.59, reinforcing the network's efficiency in connectivity despite its large scale. These insights underscore a generally efficient network with notable exceptions, potentially guiding further investigations into network optimization or targeted connectivity enhancements.

GitHub link: https://github.com/VarnikaKalani/DS210-Final