

Jaejoong Kim

Professor Kontothanassis

DS 210

14 December 2023

Final Project Report: Analysis of Social Circles in the Facebook Dataset

For my final project, I used the “Social Circles: Facebook” dataset, which encompasses a network of 4039 nodes and 88234 edges, representing mutual friendships on Facebook. The dataset offers a comprehensive look at user connections, featuring node features, circles, and ego networks. My interest in social networking and its implications in career development led me to select this dataset, particularly to explore the underlying patterns of connectivity and social reach within a large social network.

1. Project Objective

The primary goal was to analyze the Facebook dataset to uncover insights into the average distances between users, shortest paths for new connections, overall network connectivity, and unique characteristics of the most connected individuals. This exploration intended to reveal the underlying structure of social networks and their implications in real-world networking.

2. Methodology

a. Key Functions of Data Processing and Graph Construction

- The dataset was processed using Rust programming, focusing on constructing an undirected graph to represent the Facebook network. Each node corresponded to a user, and edges represented mutual friendships. While progressing through the project, I realized the potential for enhancing the analysis of the Facebook network graph by

optimizing the efficiency of average distance calculations and graph component analysis. This led to the development of several focused modules within the Rust codebase. The following key functions implemented in the codebase include:

- In the readfiles module, I implemented `read_file` for efficiently reading and processing graph data from "facebook_combined.txt". This function lays the groundwork for constructing a comprehensive graph by providing a sorted list of edges.
- The core graph construction and analysis are facilitated by the `Graph` struct and associated methods in the readfiles module. `Graph::create_undirected` is pivotal, creating an undirected graph from the list of edges, allowing for a more nuanced understanding of the network's structure.
- Significant to the project is the `print_degrees` function. It calculates and reports the degrees of the most and least connected vertices, offering insight into the network's connectivity extremes. This function highlights key nodes, identifying influencers and isolated nodes within the network.
- The `print_average_degree` function, integral to understanding the overall connectivity of the graph, computes and displays the average degree of the network. This metric is crucial for gauging the general interconnectedness of the Facebook network.
- In the component module, `count_components` plays a vital role. It assesses the graph's connectivity by counting the number of distinct connected components, shedding light on the network's segmentation or integration.

- The bfs module is critical for distance analysis. The `compute_average_distance_bfs` function, utilizing a Breadth-First Search approach, calculates the average distance between nodes. This is instrumental in understanding the 'degrees of separation' within the network.
- Further, the `subgraph` function in the bfs module is used to create smaller, manageable subgraphs. This function is vital for analyzing average distances in more focused segments of the network, allowing for a detailed understanding of local network structures.
- In the `print_average_distances` function, I employ these methods to analyze average distances not only in the entire graph but also in various subgraphs of different sizes. This multi-scale analysis provides a comprehensive view of the network's connectivity.

b. Analysis Techniques

- Breadth-First Search (BFS): Utilized to traverse the graph, starting from a node and exploring all reachable nodes. This approach helped in calculating the average distance between pairs of vertices.
- Average Distance Computation: The `'compute_average_distance_bfs'` function aggregated distances and computed the average, providing insights into the typical separation between users in the network.
- Subgraph Analysis: To understand variations in connectivity based on network size, I created subgraphs of varying node counts (100, 200, 300, 500, 1000, 2000, 3000, 4000) and calculated average distances within these subgraphs.

- Component Analysis: Employing BFS, the number of connected components in the graph was determined, revealing the level of social cohesion in the dataset.

3. Results and Observations

```

Finished dev [unoptimized + debuginfo] target(s) in 0.02s
Running `target\debug\DS210_Final_Project.exe`
Vertex with highest connectivity has 1045 connections.
Vertex with lowest connectivity has 1 connections.
Graph's average connectivity: 43.69
Total connected components in the graph: 1
Average distance among nodes in a subgraph of 100 nodes: 1.7495005
Average distance among nodes in a subgraph of 200 nodes: 2.3522105
Average distance among nodes in a subgraph of 300 nodes: 3.5051908
Average distance among nodes in a subgraph of 500 nodes: 5.358043
Average distance among nodes in a subgraph of 1000 nodes: 5.389848
Average distance among nodes in a subgraph of 2000 nodes: 5.08051
Average distance among nodes in a subgraph of 3000 nodes: 4.1217813
Average distance among nodes in a subgraph of 4000 nodes: 3.7007093
Average distance among all pairs of nodes: 3.6925068

```

- Average Distance: The average distance between pairs of vertices in the entire dataset was found to be approximately 3.69, suggesting a relatively small world phenomenon within the Facebook network.
- Subgraph Analysis: A parabolic pattern was observed in the average distances within subgraphs. The distance initially increased with the number of nodes, peaking at 500 nodes, and then decreased as the node count approached 4000.
- Component Analysis: The entire graph constituted a single connected component, indicating high interconnectivity.
- Degree Analysis: The most connected individual had 1045 connections, while the least connected had only 1. This disparity highlighted the presence of influential nodes in the network.

4. Conclusion

The analysis of the Facebook dataset revealed significant insights into the structure of social networks. The average distance of around 4 people aligns closely with the concept of 'six degrees of separation'. The subgraph analysis showed that network size influences connectivity

patterns, displaying a parabolic trend in average distances. The single-component nature of the graph underscores a high level of interconnectedness among users. These findings contribute to understanding social networks' dynamics and can inform strategies for networking and community building.

5. How to Run the Project

- Setup: Ensure Rust and Cargo are installed on the system.
- Compilation: Navigate to the project directory and run ``cargo build``.
- Execution: Execute the program using ``cargo run``.
- Output: The program outputs various statistics, including average distances, component counts, and degree information.

6. Resources

- a. Lecture notes, TAs, Homeworks, Rust Official Documents, ChatGPT
- ChatGPT Interactions

1. Implementing Breadth-First Search (BFS)

- Question: How can I implement a Breadth-First Search to find the average distance in a graph using Rust?
- Answer: Implement BFS in Rust by using a queue (`VecDeque`) to keep track of nodes to visit. Initialize the queue with a starting node, and use a vector to keep track of visited nodes and their distances. Iterate through the queue, visiting adjacent nodes, and updating their distances. To find the average distance, sum the distances of all nodes and divide by the number of nodes.

2. Counting Connected Components

- Question: What's the best way to count the number of connected components in a Rust graph structure?
- Answer: You can count connected components using a depth-first or breadth-first search algorithm. Initialize a count at zero, iterate over each vertex, and if it's not already visited, perform a DFS/BFS starting from that vertex and increment the count. Each DFS/BFS call would discover all vertices in a connected component.

3. Analyzing Graph Degrees

- Question: How do I calculate and print the degrees of vertices in a graph in Rust?
- Answer: First, create a vector to store the degree count for each vertex. Iterate over the edges and increment the degree count for both vertices on each edge. You can then iterate over the degree vector to find and print the maximum and minimum degrees, which correspond to the most and least connected vertices.

4. Calculating Average Degree

- Question: Can you explain how to compute the average degree of a graph in Rust?
- Answer: Sure, sum up all the vertex degrees by iterating over your degrees vector and then divide by the number of vertices to get the average degree. Since you've already stored the degrees in a vector, you can use the `iter` and `sum` methods to get the total degree count, then divide by the graph's vertex count.