DS210 Project Report

DOWNLOAD INSTRUCTIONS

Dataset Name: **Amazon Product Co-purchasing Network** (March 2, 2003)
URL: https://snap.stanford.edu/data/amazon0302.html

- I downloaded the file named amazon0302.txt.gz at the bottom
- Next I placed the file into the src folder of the project
    - The only place it is referenced is in the main please insure its named **amazon0302.txt**

After this it should run smoothly. I could not upload the file to github as it was to big.

Objective

In this project, I analyzed a co-purchase graph with over 334,000 products (nodes) and 925,872 co-purchase links (edges). The main goal was to calculate the Closeness Centrality, Clustering Coefficient, and identify the most central nodes in the graph to understand how the nodes are connected and which ones are most important. Since the dataset was really large, I used parallelization to make the calculations faster and more efficient.

Approach

Due to the size of the graph, processing the entire dataset wasn't feasible within the time limits. So, I decided to randomly sample 150 nodes to make the analysis more manageable. This still gave useful insights without taking up too much time or resources. The processing time for this subset was around 122.67 seconds. While this approach made the analysis easier, it had some downsides, like missing out on global patterns in the graph. This could be improved in future work by finding ways to work with the whole graph or by improving the sampling strategy.

Computational Challenges

The large size of the dataset presented some big challenges in terms of computation. Without parallelization, calculating centrality and clustering would have taken way too long. To speed up the process, I used the rayon library to parallelize the work, which helped process multiple parts of the graph at the same time and significantly reduced the total runtime.

The code

Down below I describe the main functions of my code.

Degree Centrality Computation (compute_degree_centrality)

- **Function Purpose**: This function calculates the degree centrality for each node, which tells us how many direct connections (or edges) each node has.

- **Key Components**:
    - We go through all the nodes in the graph.
    - For each node, we use graph.edges(node) to find all the connections and count them.
    - The degree centrality is simply the number of edges connected to each node.
- **Parallelization**: This computation is parallelized using rayon to speed things up, especially since the graph is large.

Closeness Centrality Computation (compute_closeness_centrality)

- **Function Purpose**: This function calculates closeness centrality, which measures how close a node is to all other nodes in the graph. I used BFS (Breadth-First Search) to approximate the shortest path lengths between nodes.
- **Key Components**:
    - I randomly selected a subset of nodes to keep the analysis efficient.
    - For each node, BFS is used to explore the shortest paths to all other reachable nodes.
    - The closeness centrality is the reciprocal of the sum of the distances to all other nodes. If a node is not connected to others, its closeness is set to 0.
- **Parallelization**: Like the degree centrality computation, I used rayon here to handle multiple nodes at once.
- **Mutex**: I used a Mutex to make sure the shared closeness_centrality data structure is safely accessed by multiple threads.

Clustering Coefficient Computation (compute_clustering_coefficient)

- **Function Purpose**: This function calculates the clustering coefficient for each node, which is a measure of how likely a node's neighbors are to be connected to each other.
- **Key Components**:
    - For each node, I count the number of triangles (i.e., pairs of neighbors that are connected to each other).
    - The clustering coefficient is calculated using the formula: C = (2 x triangles) / k(k-1) k is the degree of the node, and the numerator counts the triangles formed by the neighbors.
    - If a node has fewer than two neighbors, its clustering coefficient is set to 0.

Visualization (visualize_centrality, visualize_clustering_coefficient)

- **Function Purpose**: These functions create visual representations of the centrality and clustering coefficients of the nodes in the graph.
- **Key Components**:

    - The visualizations are saved as image files so that they can be easily viewed.
    - Before visualizing, I normalized the centrality values using the normalize_centrality_values function to scale them between 0 and 1.

Report Generation (generate_report)

- **Function Purpose**: This function generates a summary report that includes the degree centrality, closeness centrality, and clustering coefficients for all the nodes.
- **Key Components**:
    - The results are written to a text file in a readable format.
    - It lists each centrality type and its corresponding values for every node in the graph.

Normalization (normalize_centrality_values)

- **Function Purpose**: This function normalizes the centrality values so that they all fall between 0 and 1. This is helpful when visualizing the centrality, as it makes the values comparable.
- **Key Components**:
    - The maximum centrality value is found.
    - Each centrality value is divided by this maximum to normalize it.

Performance Considerations

- The program uses rayon to parallelize centrality and clustering computations, which makes the process faster, especially with large graphs like the one used in this project.
- The graph is loaded efficiently, and caching is used where necessary to avoid redundant computations, speeding up the entire process.

Overall, this project provided a way to analyze large co-purchase networks by calculating centrality measures and clustering coefficients. By using parallelization, I was able to handle the large dataset and get meaningful results in a reasonable amount of time. Although I worked with a subset of the nodes, this method still provided useful insights. In future projects, I'd like to improve the analysis by finding better ways to handle the entire graph or to improve the sampling methods to capture more global patterns.