NetworkGraphI
Twitter Social Network Graph Analysis
ALG-S2023-08

Alhossien Waly[1] and Ali Ibrahim[1]

[1]*Department of Computer Science and Engineering,*
Egypt-Japan University of Science and Technology, Alexandria, Egypt

The goal of this document is to discuss a network analysis experiment results based on the results obtained from the Twitter Social Network.
The project implementation can be found in NetworkGraphI
Also the website of the Project can be found in NetworkGraphI Website

I. PURPOSE

The Experiment aims to discover the connections between the people and themselves. In this paper, we will focus on several types of analysis including Degree Distribution, Path, Centrality, Connected Components, Clustering, Density Analysis, and Community discovery as well as visualizing the most influencing nodes in our sample.
In addition to the analysis results, we tend to analyze the algorithms that are used for the analysis.
Note that Due to ethical purposes and for the protection of privacy some examples' usernames may have characters changed with *

II. APPROACH

In this project, we used Java as the Project's programming language. We tend to collect the data using the Selenium library and Chrome Driver to scrape the official public data of Twitter users' accounts.
We stored the Obtained data set in Portable SQLite Database.
As for our last step we used JUNG Library to perform the network analysis and for the purpose of constructing the graph and visualizing graphs as well.
The project code architecture follows the command design pattern for the sake of ease of control in both the Data collection stage and Network analysis stage.

III. DATA COLLECTION

At first, we collected the data of Twitter accounts putting into consideration the ethical and legal manners with the use and publication of the information we've collected. All our privacy terms and ease of use are discussed in NetworkGraphI Website. We started the collection by putting an initial node then we started by gathering information about all the followers and followings of that parent node.
For some important nodes that have more than 2000 followers or followings we limited the data collection in that specific case to get the first 2000 followers or followings for time consumption purposes as well as not focusing the graph to a single point connection.
After the retrieval of the data, we constructed a temporary graph to eliminate all errors that might happen during the collection of data. after the construction of the graph, we traverse and discover the graph using breadth-first traversal to update the database with our graph info before attempting analysis.

IV. DEGREE DISTRIBUTION ANALYSIS

the degree of a node is the number of points connected to it. the degree distribution is the probability distribution of all nodes in the graph.
1- iterate all over all nodes in the graph using Breadth-First-Search.
2- get every node degree by counting how many elements are connected to it.
3- once done all nodes calculate the value of degree distribution.

here is the algorithm I followed to find out degree distribution:
Algorithm: Degree Distribution Analysis using X-Chart

Input: Graph G
1. Initialize an empty map degree distribution to store the degree-frequency pairs.
2. For each vertex v in G:
2.1 Calculate the degree of vertex v and store it in the variable degree.
2.2 Increment the frequency of degree in the degree distribution map.
3. Create an XY-Chart object named chart.
3.1 Set the width and height of the chart.
3.2 Set the title of the chart as "Degree Distribution".
3.3 Set the x-axis title as "Degree" and the y-axis title as "Frequency".
4. Customize the style of the chart:
4.1 Set the default series render style to Line.
5. Create arrays of degrees and frequencies of size equal to the number of entries in degree distribution.

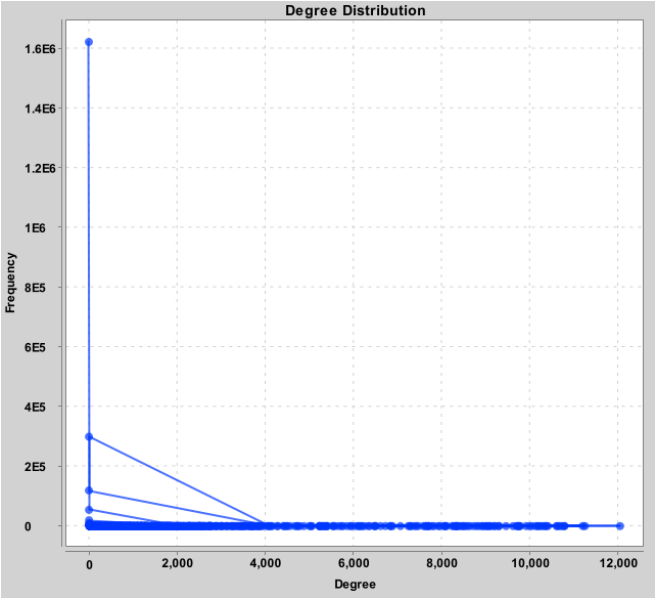

FIG. 1. degree distribution

6. Initialize an index variable to 0.
7. For each entry (degree, frequency) in the degree distribution map:
7.1 Store degree at the index position in the degrees array.
7.2 Store frequency at the index position in the frequencies array.
7.3 Increment the index variable.
8. Add the series "Degree Distribution" to the chart:
8.1 Use degrees and frequency arrays as arguments to represent x and y values.
9. Display the chart using a SwingWrapper.
Output: A chart displaying the degree distribution of the graph.

the degree distribution result of our data:

V. PATH ANALYSIS

an example of path analysis is finding the shortest path to find it using Dijkstra's Algorithm: Dijkstra's algorithm is used to find the shortest path between a source vertex and all other vertices in a weighted graph. how it works: with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

here is the algorithm I followed to find out the shortest path:

Algorithm: Shortest Path Calculation using Dijkstra's Algorithm and JUNG
Input: Graph G, start vertex start, end vertex end

1. Create a DijkstraShortestPath object named shortest path, passing the graph G as a parameter.

2. Use the shortest-path object to find the shortest path between the start vertex and the end vertex:
2.1 Call the getPath(start, end) method of the shortest path object, storing the result in a list named path.
3. Output the shortest path:
3.1 Print the elements of the path list.

Output: The shortest path between the start and end vertices.

we apply shortest path Analysis for random vertex and we found the result as flow:
the shortest path between Alhoss***_W**y Account and ea*ma****arab Account is as follows:
1-sa***r*__. 2-Si***ch. 3-AlsisiOfficial.

VI. CENTRALITY ANALYSIS

A. Degree centrality

Degree centrality measures the number of connections or edges that a node has in the network.

B. Betweenness centrality

Betweenness centrality measures the extent to which a node lies on the shortest paths between other pairs of nodes in the network. Nodes with high betweenness centrality act as bridges or intermediaries between different parts of the network.

C. Closeness Centrality

Closeness centrality measures the average distance between a node and all other nodes in the network. Nodes with high closeness centrality are typically more reachable and have shorter paths to other nodes.
Algorithm: Centrality Analysis using JUNG

Input: Graph G, vertex vertax

1. Calculate the degree centrality of the vertex in the graph:
1.1 Call the vertex degree() method, passing the graph G and the vertex as parameters.
1.2 Output the degree centrality value.
2. Compute the betweenness centrality of the vertex using the BetweennessCentrality measure:
2.1 Create a BetweennessCentrality object named

betweenness, passing the graph G as a parameter.
2.2 Set the removeRankScoresOnFinalize property of the betweenness object to false.
2.3 Evaluate the betweenness centrality.
2.4 Retrieve the betweenness centrality score of the vertex using the getVertexRankScore() method.
2.5 Output the betweenness centrality value.
3. Compute the closeness centrality of the vertex using the closeness centrality measure:
3.1 Create a ClosenessCentrality object named closeness, passing the graph G as a parameter.
3.2 Compute the closeness centrality.
3.3 Retrieve the closeness centrality score of the vertex using the getVertexScore() method.
3.4 Output the closeness centrality value.

Output: Degree centrality, betweenness centrality and closeness centrality values of the vertex.

we apply Centrality Analysis for random vertex and we found the result as flow:
1- ChelseaFC Account:

- a- Degree centrality: 10054.
- b- Betweenness centrality: 0.0.
- c- Closeness centrality: 1.0.

2- R*C**rab Account:

- a- Degree centrality: 9025.
- b- Betweenness centrality: 0.0.
- c- Closeness centrality: 0.9997784669915818.

VII. CONNECTED COMPONENTS ANALYSIS

Connected components analysis is a technique used to identify and analyze the connected subgraphs within a larger graph. A connected component is a set of nodes within a graph where each node is reachable from every other node in the component.

how it works:
1- Start with an empty list or data structure to store the connected components.
2- Iterate through each node in the graph.
3-For each unvisited node, perform a depth-first search (DFS) or breadth-first search (BFS) starting from that node. During the search, mark each visited node as visited to avoid revisiting it.
4- When the search finishes, you have discovered a connected component. Add the component to the list of connected components.
5- The final list of connected components represents the disjoint subsets of the graph.

here is the algorithm I followed to find out Connected Components Analysis:

Algorithm: Connected Components Analysis using JUNG
Input: Graph G, vertex vertax
1. Create a WeakComponentClusterer object named clustered.
2. Use the clustered object to find the connected components in graph G:
2.1 Call the transform() method of the clustered object, passing the graph G as a parameter.
2.2 Store the result in a set named components.
3. Output the number of connected components found:
3.1 Print "Found " followed by the size of the components set, followed by " connected component(s)".
4. Output the nodes in each connected component: 4.1 Iterate over each set component in the components set.
4.2 For each component, print the nodes in the component.

Output: The number of connected components found and the nodes in each connected component.

we apply connected components analysis for random vertex and we found the result as flow:
1- 4331 connected components to RMCF**ab account.
2- 4331 connected components to AnasB**hash account.

VIII. CLUSTERING COEFFICIENTS

Clustering Coefficients are measures used to quantify the extent to which nodes in a graph tend to form clusters or communities. The clustering coefficient of a node is a ratio that measures the proportion of connections between its neighbors. It indicates how likely the neighbors of a node are connected to each other. There are different variations of clustering coefficients, but two commonly used ones are:
1- Local Clustering Coefficient: The local clustering coefficient of a node measures the proportion of connections between its neighbors. It is calculated as the number of actual connections between the neighbors divided by the number of possible connections.
2- Global Clustering Coefficient: The global clustering coefficient of a graph measures the overall clustering tendency of the network. It is calculated as the average of the local clustering coefficients of all nodes in the graph.
Calculating the clustering coefficient of a node involves counting the number of connections between its neighbors and determining the total number of possible connections. The formula for the local clustering coefficient of a node "v" is:

local clustering coefficient(v) =
2 * numberOfConnectionsBetweenNeighbors(v) / (degree(v) * (degree(v) - 1))

The clustering coefficient can help identify nodes or regions in a graph that are highly interconnected, indicating the presence of clusters or communities. It is often used in network analysis, social network analysis, and community detection algorithms to understand the structure and organization of complex networks.

here is the algorithm I followed to find out Clustering Coefficients:

Algorithm: Clustering Coefficients Analysis using JUNG
Input: Graph G
1. For each node v in the graph G:
1.1 Calculate the degree k of node v using the degree() method of the graph.
1.2 Initialize a variable triangle to 0.
1.3 Iterate over each pair of neighbors (u, w) of node v:
1.3.1 Check if nodes u and w are neighbors using the isNeighbor() method of the graph.
1.3.2 If nodes u and w are neighbors, increment the triangles variable.
1.4 Calculate the clustering coefficient cc of node v:
1.4.1 If k is greater than 1, compute cc as triangles divided by (k * (k - 1) / 2).
1.4.2 Output the clustering coefficient of node v.
2. Initialize variable triangles to 0. Iterate over each pair of nodes (u, v) in the graph G:
2.1 Check if nodes u and v are neighbors using the isNeighbor() method of the graph. 2.2 If nodes u and v are neighbors, increment the triangles variable.
3. Calculate the global clustering coefficient of graph G:
3.1 Get the edge count k of the graph using the getEdgeCount() method.
3.2 Calculate the global clustering coefficient global CC as triangles divided by (k * (k - 1) / 2).
3.3 Output the global clustering coefficient.
Output: The clustering coefficient of each node in the graph and the global clustering coefficient.

The Global clustering coefficient of our data: 0.0017061409184009045

IX. DENSITY ANALYSIS

Density analysis is a measure used to quantify the compactness or connectedness of a graph. It provides insight into how closely the nodes in a graph are interconnected and can indicate the level of cohesion within the network. The density of a graph is calculated by dividing the actual number of edges in the graph by the maximum possible number of edges. It is typically expressed as a value between 0 and 1, where 0 represents no edges and 1 represents a fully connected graph. The formula to calculate the density of an undirected graph with 'n' nodes is:
density = (2 * abs(E)) / (n * (n - 1)) Where:
abs(E) is the number of edges in the graph. 'n' is the number of nodes in the graph.
For directed graphs, the formula is slightly different:
density = abs(E) / (n * (n - 1))
Density analysis provides a measure of how dense or sparse a graph is. A higher density indicates a higher level of connectivity and a greater likelihood of information flow or interaction between nodes. On the other hand, a lower density suggests a more sparse or disconnected network. Density analysis can be useful in various applications, such as social network analysis, transportation network analysis, or analyzing the robustness of a network. It helps identify the overall connectedness of a graph and provides insights into its structure and potential information flow.

here is the algorithm I followed to find out Density Analysis:

Algorithm: Density Analysis using JUNG
Input: Graph G
1. Get the number of vertices V in the graph G using the getVertexCount() method.
2. Get the number of edges E in the graph G using the getEdgeCount() method.
3. Calculate the density of the graph:
3.1 Initialize a variable density to 0.0.
3.2 Calculate the density as 2 * E / (V * (V - 1)).
4. Output the density of the graph.
Output: The density of the graph.

The Density of our data graph: 0.0018742940535781084

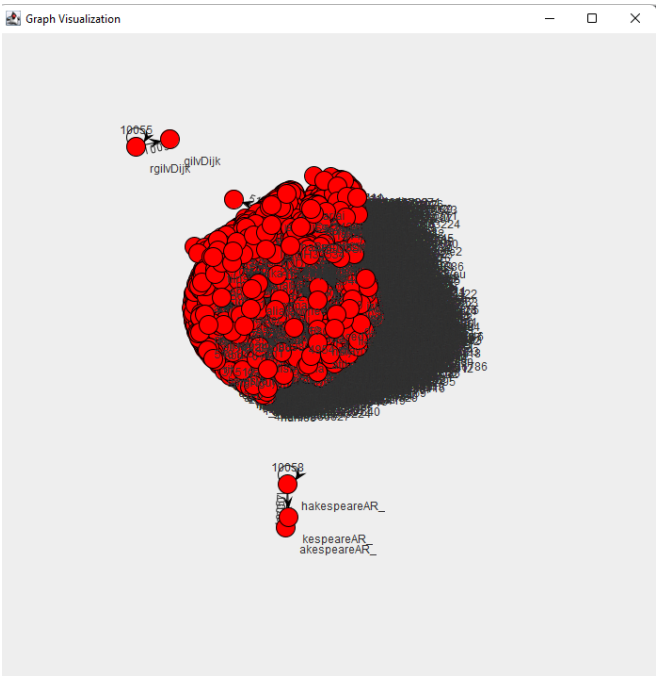

FIG. 2. ChelseaFC community

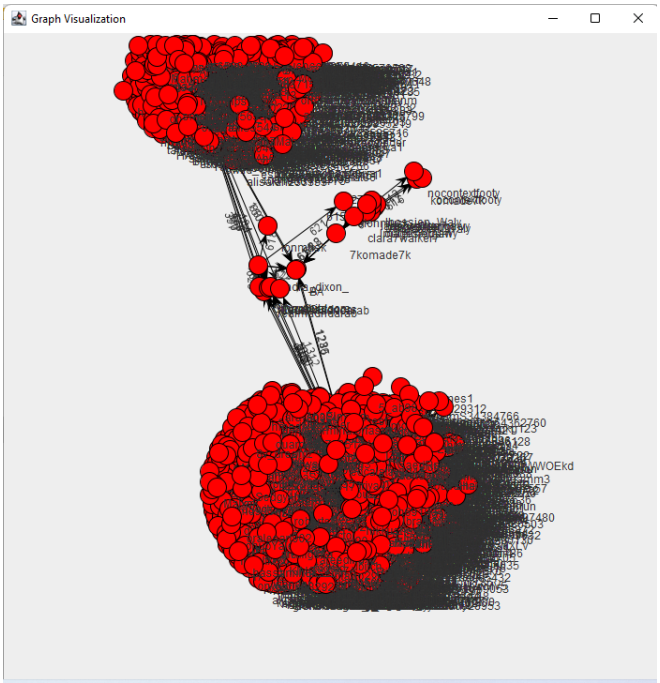

FIG. 3. AlsisiOfficial community

X. COMMUNITY DISCOVERY

Community discovery, also known as community detection or graph clustering, is the process of identifying groups or communities within a network. It aims to partition the nodes of a graph into cohesive subgroups where nodes within the same group have stronger connections among themselves compared to nodes in different groups. Community discovery is a fundamental task in network analysis and has applications in various fields, including social network analysis, biological networks, recommendation systems, and information retrieval. By uncovering the underlying community structure, it helps understand the organization, behavior, and interactions within a complex network. There are several algorithms and methods available for community discovery in graphs. Some commonly used ones include:
Louvain Algorithm: The Louvain algorithm is a popular and efficient method for community detection. It optimizes a quality function known as modularity to iteratively merge and refine communities until an optimal partition is obtained.

here is the algorithm I followed to find out Community discovery:

Algorithm: Community Discovery using JUNG
Input: Graph G
1. Initialize an empty set to store the detected communities.
2. For each vertex in the graph G:
2.1 Create a new set to represent a community.
2.2 Add the vertex to the community.
2.3 Create a set of neighbors for the vertex.
2.4 While the set of neighbors is not empty:
2.4.1 Get a neighbor from the set of neighbors.
2.4.2 Add the neighbor to the community.
2.4.3 Add the neighbors of the neighbor to the set of neighbors.
2.4.4 Remove the added neighbors from the community.
2.5 Add the community to the set of detected communities.
3. Output the detected communities.
Output: The detected communities in the graph. some detected communities in our data:

XI. NETWORK VISUALIZATION

Network visualization is the process of representing and displaying a graph or network in a visual form. It aims to visually convey the relationships and connections between nodes (vertices) and edges (links) in a graph, allowing for easier understanding and analysis of the network structure. Network visualization can be beneficial in various fields, including social network analysis, biological network analysis, transportation network analysis, and information visualization. It helps uncover patterns, clusters, and other structural properties of the network that may not be immediately apparent in a raw data form.

here is the algorithm I followed to Network visual-

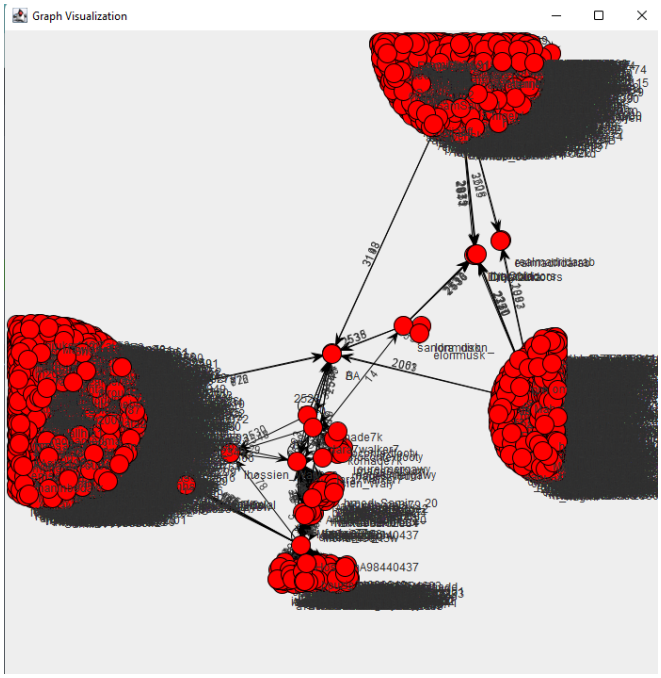

FIG. 4. Network Visualization

ization.
Algorithm: Network Visualization using JUNG
Input: Graph G
1. Create a layout for the graph using an appropriate layout algorithm (e.g., FRLayout).
2. Set the size of the layout.
3. Create a VisualizationViewer to display the graph.
4. Set the preferred size of the VisualizationViewer.
5. Set the vertex and edge labels using ToStringLabeller.
6. Set the vertex and edge shapes (e.g., using Ellipse2D.Double for vertices and EdgeShape.Line for edges).
7. Set up mouse interaction (e.g., using DefaultModalGraphMouse).
8. Create a JFrame to hold the visualization. 9. Set the JFrame's title.
10. Set the default close operation of the JFrame.
11. Add the VisualizationViewer to the JFrame's content pane.
12. Pack the JFrame to set the appropriate size.
13. Set the JFrame to be visible.
Output: A visual representation of the graph.
Network Visualization of some data: