Network and Graph Analysis: Analyzing Relationships Between Actors Based on Their Shared Movies Group: ALG-S2023-09

Gasser Amr[1], Rewan Yehia[1], Bassant Tarek[1], and Ahmed Ali[1]

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

A network analysis was presented of actors based on the movies they have acted in together. By analyzing the relationships between actors, we can gain insights into the structure and dynamics of the film industry, identify closely-knit groups of actors, and find influential actors within the network

1. INTRODUCTION

The film industry is a complex ecosystem of actors, directors, producers, and other professionals working together to create cinematic experiences. Understanding the relationships and connections between these individuals can provide valuable insights into the industry's structure, trends, and collaboration patterns.

A. Objectives

- To create an actor-network graph representing the connections between actors based on their movie collaborations.
- To analyze the graph's properties, such as degree distribution, shortest path lengths, clustering, communities, and centrality Measure.
- To identify and interpret patterns in the actor-network that can inform our understanding of the film industry.

To achieve these objectives, we collected a comprehensive dataset using the TMDb API and web scraping techniques on IMDb. Our dataset includes English movies released between 2000 and 2023.

In this report, we describe the data collection and preprocessing steps, detail the creation of the actor-network graph, and present the results of our network and path analysis. Additionally, we provide visualizations of the graph's properties and discuss the key insights gained from our findings. By examining the actor-network, we aim to contribute to a deeper understanding of the film industry and the collaborative dynamics that shape it.

2. DATA COLLECTION AND PREPOSSESSING

Our dataset was collected from two primary sources: The Movie Database (TMDb) API web scraping IMDb. These sources allowed us to gather a comprehensive list of English movies from 2000 to 2023, along with their associated cast, directors, ratings, and release years. The data collection process involved the following steps:

1. TMDb API: We used the TMDb API to retrieve movie information, including movie titles, ratings, and release years. The API also provided us with a list of cast members and directors for each movie.
2. IMDb Web Scraping: To supplement the data obtained from the TMDb API, we scraped IMDb for additional movie and actor information. This step enabled us to ensure the completeness and accuracy of our dataset.

After collecting the data, we performed several preprocessing steps to clean, filter, and transform the raw data into a suitable format for network analysis:

- Data Cleaning: We removed any duplicate entries, corrected spelling errors, and standardized the formatting of movie titles, actor names, and other text data.
- Data Filtering: We filtered out movies with incomplete information, such as missing cast or director data. Additionally, we excluded movies with very low ratings or limited relevance to our analysis.
- Data Transformation: We transformed the data into a structured format that allowed us to create the actor-network graph. Specifically, we created a list of unique actors and movies.

With our cleaned and preprocessed data, we proceeded to create the actor network graph

3. NETWORK CREATION

Our approach for creating the actor network graph is based on the provided Python code, which takes a DataFrame of movie information as input and constructs a network graph using the NetworkX library. The nodes in the graph represent actors, and the edges between nodes represent the relationships between actors based on the movies they have acted in together. Each edge is weighted based on the number of movies the connected actors have collaborated on, and the edge attributes also contain a list of movie titles.

The main steps of our methodology are as follows:

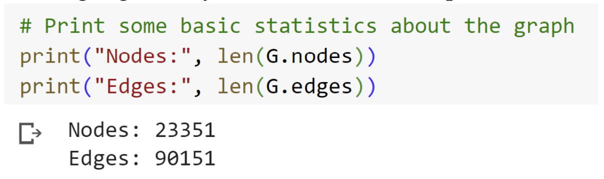
- Extract unique actor names from the input DataFrame.
- Create an undirected graph and add nodes for actors.
- Count co-appearances of actors in movies and add weighted edges accordingly.
- Add movie titles to the edge attributes.
- Remove self-loop edges, empty or missing labels, and isolated nodes.

4. DESCRIPTIVE STATISTICS

We provide an in-depth analysis of the basic properties and characteristics of the network under study. We calculate key network properties, such as the number of nodes and edges, average degree, density, and diameter, and discuss the implications of these statistics for the overall structure and connectivity of the network. Understanding these properties is essential for gaining insights into the network's behavior and functioning.

A. Number of Nodes and Edges

The number of nodes (N) in a network represents the total number of entities, while the number of edges (E) represents the connections between these entities. A higher number of nodes and edges generally indicates a more complex network.

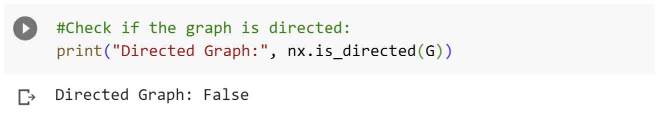


```
# Print some basic statistics about the graph
print("Nodes:", len(G.nodes))
print("Edges:", len(G.edges))
```

```
Nodes: 23351
Edges: 90151
```

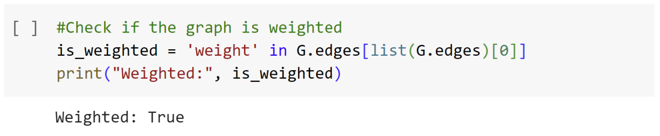
B. Network Type

To analyze network type we performed the following:



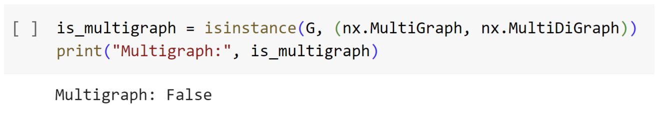
```
#Check if the graph is directed:
print("Directed Graph:", nx.is_directed(G))
```

```
Directed Graph: False
```



```
[ ] #Check if the graph is weighted
    is_weighted = 'weight' in G.edges[list(G.edges)[0]]
    print("Weighted:", is_weighted)
```

```
Weighted: True
```

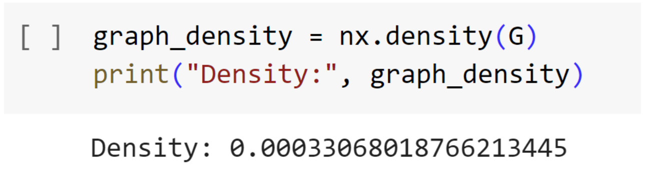


```
[ ] is_multigraph = isinstance(G, (nx.MultiGraph, nx.MultiDiGraph))
    print("Multigraph:", is_multigraph)
```

```
Multigraph: False
```

C. Density

The density of a network is the ratio of the number of observed edges to the maximum possible number of edges. It ranges between 0 (no edges) and 1 (complete graph). A higher density indicates a more connected network.

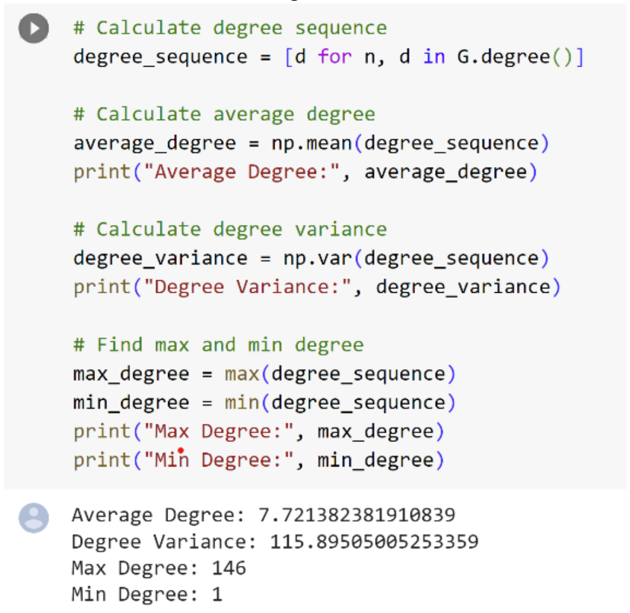


```
[ ] graph_density = nx.density(G)
    print("Density:", graph_density)
```

```
Density: 0.00033068018766213445
```

D. Degree Distribution

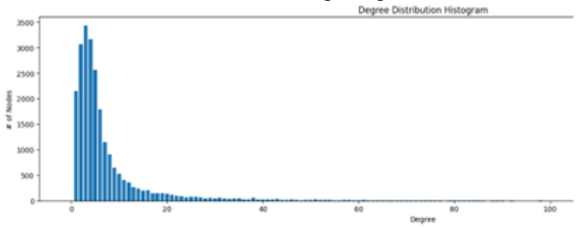
We will analyze the degree distribution of the network, which can provide insights into the organization and topology of the network.
We calculated for an average degree, and degree variance and found the Max and Min degree.



```
# Calculate degree sequence
degree_sequence = [d for n, d in G.degree()]

# Calculate average degree
average_degree = np.mean(degree_sequence)
print("Average Degree:", average_degree)

# Calculate degree variance
degree_variance = np.var(degree_sequence)
print("Degree Variance:", degree_variance)

# Find max and min degree
max_degree = max(degree_sequence)
min_degree = min(degree_sequence)
print("Max Degree:", max_degree)
print("Min Degree:", min_degree)
```

```
Average Degree: 7.721382381910839
Degree Variance: 115.89505005253359
Max Degree: 146
Min Degree: 1
```

Then, it was ploted as the Degree Distribution using *plt.hist*(), which resulted in the following diagram:

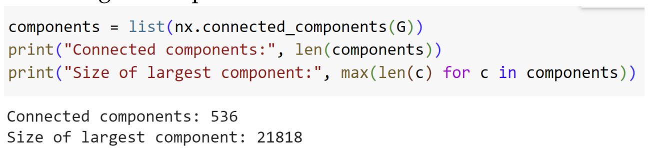


D.1. Algorithm & Complexity

the complexity of the algorithm used is O(n + m) where n is the number of nudes and m is the unique degree values

E. Connected Components

Connected components in a network are subgraphs in which every pair of nodes is connected by at least one path. Analyzing connected components can provide insights into the cohesiveness and fragmentation of the network. In this section, we will analyze the connected components of the network and discuss the implications of these findings.
We calculated the number of connected components and the size of the largest component:



```
components = list(nx.connected_components(G))
print("Connected components:", len(components))
print("Size of largest component:", max(len(c) for c in components))
```

```
Connected components: 536
Size of largest component: 21818
```

E.1. Algorithm & Complexity

The $nx.connected_components()$ function in the NetworkX library finds the connected components of an undirected graph using a depth-first search (DFS) algorithm.

The algorithm explores the entire graph using DFS, marking nodes as visited and grouping them into connected components. The time complexity of this algorithm is $O(n + m)$, where n is the number of nodes and m is the number of edges in the graph.

F. Implications of Network Statistics for Overall Structure and Connectivity

The network is undirected and weighted, which means that connections between nodes have associated weights representing the strength of the relationship. The network is not a multigraph, so there are no parallel edges between any pair of nodes.

The density of the network is relatively low (0.00033068018766213445), indicating that the network is sparse and that nodes are not highly connected. This might suggest that the network exhibits a modular structure, where nodes tend to form tight-knit communities with relatively few connections between communities.

There are 536 connected components in the network, with the largest component containing 21,818 nodes. This means that the majority of nodes are part of the largest connected component, indicating a giant component in the network. The presence of a giant component can be an indication of the small-world property, where most nodes can be reached from any other node by a small number of steps.

The average degree of nodes in the network is 7.72, with a degree variance of 115.90. The large variance suggests that the network likely has a heterogeneous degree distribution, with some nodes having many more connections than others. The maximum degree is 146, which is significantly higher than the average, indicating the presence of hubs or highly connected nodes in the network. These hubs can play an important role in information flow and network robustness.

The minimum degree is 1, indicating that there are nodes with only one connection. These nodes can be considered peripheral nodes, which might have limited influence on the overall network. In summary, the network exhibits a sparse, modular structure with a giant component, hubs, and peripheral nodes. These properties have implications for the overall connectivity, information flow, and robustness of the network.

5. PATH ANALYSIS

In this study, we performed path analysis to find the shortest paths between pairs of actors in a network. The network represents actors as nodes and the connections between them are based on their co-starring in movies. We utilized the NetworkX library, a widely-used Python library for the creation, manipulation, and study of complex networks.

A. Method

We employed the NetworkX library to construct the network of actors. The core of our analysis was finding the shortest paths between pairs of actors using various algorithms provided by NetworkX.

B. Shortest Path Algorithms

- Shortest Path: We first used the shortest _path function to find the shortest path between two actors, for eg. Tom Cruise and Meryl Streep. This function uses Dijkstra's algorithm by default if the graph is weighted, or Breadth-First Search (BFS) if the graph is unweighted. In our case, the graph is unweighted, so BFS was used. The time complexity of BFS is $O(V + E)$, where V is the number of nodes (actors) and E is the number of edges (connections).

```
Shortest path between Tom Cruise and Meryl Streep : ['Tom Cruise', 'Meryl Streep']
```

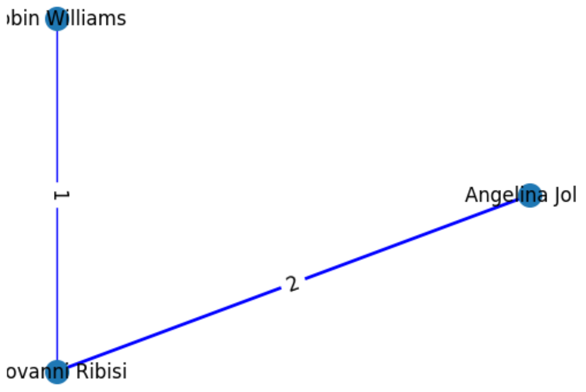
- All Shortest Paths: Next, we used the all_shortest_paths function to find all shortest paths between for eg. Toni Collette and Tom Cruise. This function also uses BFS, maintaining the same time complexity as the shortest_path function.

```
['Toni Collette', 'Jesse Plemons', 'Tom Cruise']
['Toni Collette', 'Dakota Fanning', 'Tom Cruise']
['Toni Collette', 'Colin Farrell', 'Tom Cruise']
['Toni Collette', 'Philip Seymour Hoffman', 'Tom Cruise']
['Toni Collette', 'Cameron Diaz', 'Tom Cruise']
['Toni Collette', 'Alec Baldwin', 'Tom Cruise']
```

- Dijkstra's Algorithm: Lastly, we employed the dijkstra _path function to find the shortest path between Angelina Jolie and Robin Williams. This function uses Dijkstra's algorithm for weighted graphs. The time complexity of Dijkstra's algorithm is $O(V + E)$ for a sparse graph when implemented using a priority queue.

C. Visualization

We visualized the subgraph of relevant actors and their connections using the NetworkX drawing functions. We utilized a circular layout for the nodes and displayed edge weights as edge labels.



D. Interpretation

The shortest path between two actors represents the minimum number of co-starring connections required to "link" the two actors. This can be interpreted as a measure of the "distance" between them in terms of their professional collaborations.

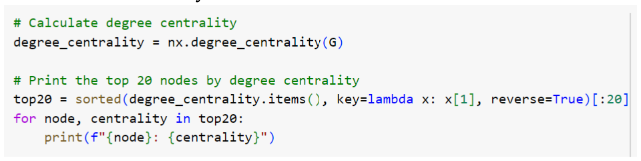
6. CENTRALITY MEASURES

To identify the most central and influential actors in the network, you can use various centrality measures, which quantify the importance of nodes within the network. Here are some common centrality measures that can help you identify key actors:

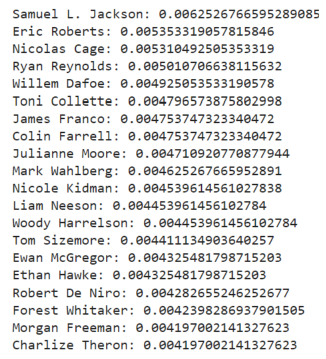
A. Degree Centrality

Degree centrality is the simplest measure of centrality, which is calculated by counting the number of edges connected to a node. Actors with a high degree of centrality have collaborated with many other actors, indicating that they are well-connected in the network. Key actors with a high degree of centrality can be seen as popular, sought-after, or versatile, as they have worked with a diverse range of actors throughout their careers.

I used networkx's *.degree_centrality*() function to retrieve the network graph's degree centrality. Then, I printed the 20 actors with the highest degree of centrality. Degree centrality is based on the number of edges the node has. Therefore, it shows which actors have the most direct movie connections, meaning the movies they are known for that are in common with most other actors' movies they are known for.

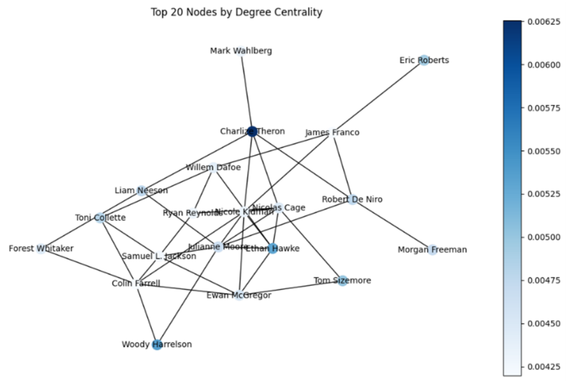


```
# Calculate degree centrality
degree_centrality = nx.degree_centrality(G)

# Print the top 20 nodes by degree centrality
top20 = sorted(degree_centrality.items(), key=lambda x: x[1], reverse=True)[:20]
for node, centrality in top20:
    print(f"{node}: {centrality}")
```



```
Samuel L. Jackson: 0.0062526766595289085
Eric Roberts: 0.005353319057815846
Nicolas Cage: 0.005310492505353319
Ryan Reynolds: 0.005010706638115632
Willem Dafoe: 0.004925053533190578
Toni Collette: 0.004796573875802998
James Franco: 0.004753747323340472
Colin Farrell: 0.004753747323340472
Julianne Moore: 0.004710920770877944
Mark Wahlberg: 0.004625267665952891
Nicole Kidman: 0.004539614561027838
Liam Neeson: 0.004453961456102784
Woody Harrelson: 0.004453961456102784
Tom Sizemore: 0.004411134903640257
Ewan McGregor: 0.004325481798715203
Ethan Hawke: 0.004325481798715203
Robert De Niro: 0.004282655246252677
Forest Whitaker: 0.0042398286937901505
Morgan Freeman: 0.004197002141327623
Charlize Theron: 0.004197002141327623
```

Finally, I created a graph to show the top20 nodes using networkx's *.spring_layout*() and *.draw*() I displayed it using matplotlib's *.show*().



I repeated the centrality calculations and displays for the rest of the centrality calculations.

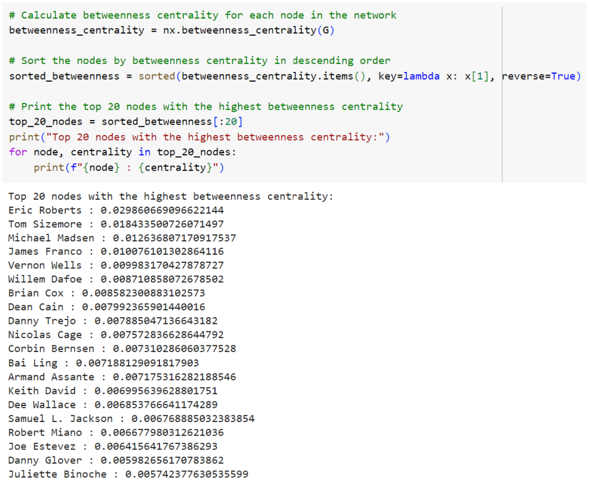
A.1. Algorithm & Complexity

The algorithm used by nx.degree_centrality is quite straightforward. It calculates the degree centrality for each node in the graph by dividing the degree of the node (the number of edges connected to the node) by the maximum possible degree (which is n-1, where n is the total number of nodes in the graph).

The time complexity of nx.degree_centrality depends on the underlying data structure used to represent the graph. For sparse graphs (in our case), the time complexity is O(n + m), where n is the number of nodes and m is the number of edges.

B. Betweenness Centrality

Betweenness centrality measures how often a node lies on the shortest path between other nodes in the network. Actors with high betweenness centrality act as "bridges" in the network, connecting different communities or clusters. These actors may be influential in the sense that they bring together diverse groups of actors and have the potential to facilitate new collaborations.



```
# Calculate betweenness centrality for each node in the network
betweenness_centrality = nx.betweenness_centrality(G)

# Sort the nodes by betweenness centrality in descending order
sorted_betweenness = sorted(betweenness_centrality.items(), key=lambda x: x[1], reverse=True)

# Print the top 20 nodes with the highest betweenness centrality
top_20_nodes = sorted_betweenness[:20]
print("Top 20 nodes with the highest betweenness centrality:")
for node, centrality in top_20_nodes:
    print(f"{node} : {centrality}")
```

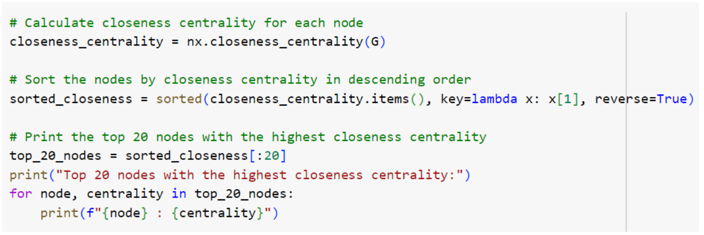
```
Top 20 nodes with the highest betweenness centrality:
Eric Roberts : 0.029860669096622144
Tom Sizemore : 0.018433500726071497
Michael Madsen : 0.012636807170917537
James Franco : 0.010076101302864116
Vernon Wells : 0.009983170427878727
Willem Dafoe : 0.008710858072678502
Brian Cox : 0.008582300883102573
Dean Cain : 0.007992365901440016
Danny Trejo : 0.007885047136643182
Nicolas Cage : 0.007572836628644792
Corbin Bernsen : 0.007310286060377528
Bai Ling : 0.007188129091817903
Armand Assante : 0.007175316282188546
Keith David : 0.006995639628801751
Dee Wallace : 0.006853766641174289
Samuel L. Jackson : 0.006768885032383854
Robert Miano : 0.006677980312621036
Joe Estevez : 0.006415641767386293
Danny Glover : 0.005982656170783862
Juliette Binoche : 0.005742377630535599
```

B.1. Algorithm & Complexity

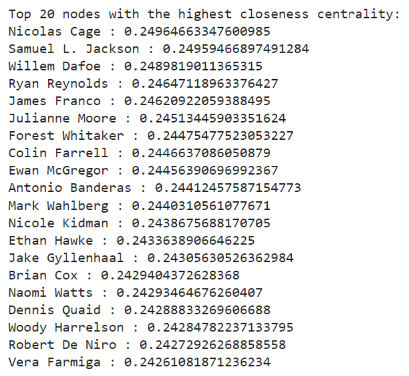
there are faster algorithms available for approximating the betweenness centrality, such as Brandes' algorithm, which has a time complexity of O(N * E) in networks. These algorithms exploit properties of the network structure to compute an approximation of the betweenness centrality with reduced computational cost.

C. Closeness Centrality

Closeness centrality measures the average shortest path length from a node to all other nodes in the network. Actors with high closeness centrality have short connection paths to other actors, suggesting that they can easily collaborate or communicate with others in the network. This can be an indicator of an actor's ability to access information and resources within the industry quickly.



```
# Calculate closeness centrality for each node
closeness_centrality = nx.closeness_centrality(G)

# Sort the nodes by closeness centrality in descending order
sorted_closeness = sorted(closeness_centrality.items(), key=lambda x: x[1], reverse=True)

# Print the top 20 nodes with the highest closeness centrality
top_20_nodes = sorted_closeness[:20]
print("Top 20 nodes with the highest closeness centrality:")
for node, centrality in top_20_nodes:
    print(f"{node} : {centrality}")
```



```
Top 20 nodes with the highest closeness centrality:
Nicolas Cage : 0.24964663347600985
Samuel L. Jackson : 0.24959466897491284
Willem Dafoe : 0.2489819011365315
Ryan Reynolds : 0.24647118963376427
James Franco : 0.24620922059388495
Julianne Moore : 0.24513445903351624
Forest Whitaker : 0.24475477523053227
Colin Farrell : 0.2446637086050879
Ewan McGregor : 0.24456390696992367
Antonio Banderas : 0.24412457587154773
Mark Wahlberg : 0.2440310561077671
Nicole Kidman : 0.2438675688170705
Ethan Hawke : 0.2433638906646225
Jake Gyllenhaal : 0.24305630526362984
Brian Cox : 0.2429404372628368
Naomi Watts : 0.24293464676260407
Dennis Quaid : 0.24288833269606688
Woody Harrelson : 0.24284782237133795
Robert De Niro : 0.24272926268858558
Vera Farmiga : 0.24261081871236234
```

C.1. Algorithm & Complexity

Dijkstra's algorithm is being used to calculate the shortest path lengths in the closeness centrality algorithm. Dijkstra's algorithm is an efficient algorithm for finding the shortest paths between nodes in a graph with non-negative edge weights. It iteratively explores the graph, starting from the source node, and updates the shortest path distances to reach each node.

D. Eigenvector Centrality

Eigenvector centrality takes into account not only the number of connections a node has but also the quality of those connections. Actors with high eigenvector centrality are connected to other well-connected actors, which may indicate a strong influence within the industry. These actors are often part of a "core" group of influential actors who frequently work together.

```
# Calculate eigenvector centrality
centrality_eigenvector = nx.eigenvector_centrality(G)

# Print top 20 nodes by eigenvector centrality
top_nodes = sorted(centrality_eigenvector.items(), key=lambda x: x[1], reverse=True)[:20]
for node, centrality in top_nodes:
    print(f"{node}: {centrality}")
```

```
Samuel L. Jackson: 0.1006177518918609
Julianne Moore: 0.08959700245869862
Colin Farrell: 0.08770879075431622
Mark Wahlberg: 0.08761025625354942
Nicole Kidman: 0.08252149325369819
Matt Damon: 0.08101050503498622
Ryan Reynolds: 0.08046917221201359
Cate Blanchett: 0.07866259693628914
Christian Bale: 0.07777995583920251
Nicolas Cage: 0.075863153568224
Ewan McGregor: 0.07576479408517262
Woody Harrelson: 0.07388065166144815
Robert De Niro: 0.07258468096585384
Charlize Theron: 0.07135768287936077
Jude Law: 0.07078951972238526
Scarlett Johansson: 0.07006519691915881
Jake Gyllenhaal: 0.06948578919909681
Anthony Hopkins: 0.06931177674806138
Naomi Watts: 0.06927389122714231
Susan Sarandon: 0.06849594568980011
```

D.1. Algorithm & Complexity

The Algorithm being used is the Power Method algorithm which is commonly used to compute the principal eigenvector, which corresponds to the eigenvector with the largest eigenvalue. The power method is an iterative algorithm that repeatedly multiplies the adjacency matrix by a vector and normalizes the result until convergence is reached. The complexity of the power method algorithm depends on the size of the network and the convergence criteria. In general, the algorithm has a time complexity of $O(nm)$, where n is the number of nodes and m is the number of edges in the graph.

7. CLUSTERING COEFFICIENT

In this section, we will discuss the calculation of clustering coefficients to assess the presence of cliques or tightly-knit groups within the network. The clustering coefficient is a useful metric to understand the degree of interconnectedness among the nodes in a network.

We used the function $nx.clustering()$, which is based on an algorithm known as the "local clustering algorithm" or "triangle counting algorithm."

The graph which was produced is an unweighted graphs. To calculate the clustering of a node:

$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)}$,

where $T(u)$ is the number of triangles through node u and $deg(u)$ is the degree of u.

Since we analyzed that our graph is a sparse graphs where the number of edges is much smaller than the number of nodes squared, the algorithm's time complexity is typically $O(n^2)$, where n is the number of nodes in the graph.

A. Calculating the Average Clustering Coefficient

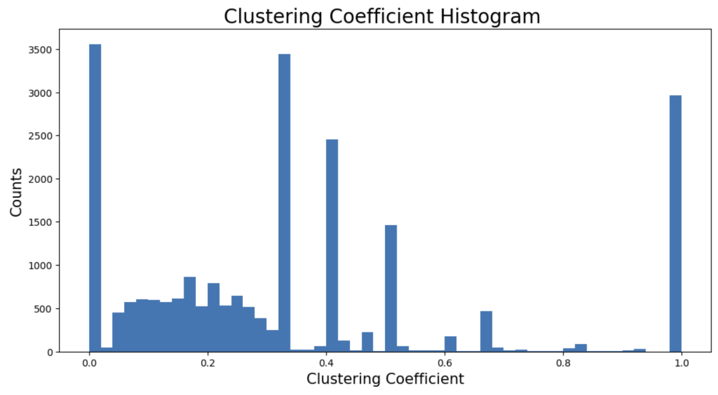
To assess the presence of cliques or tightly-knit groups within the network, we can calculate the average clustering coefficient for the entire graph. To do this, we will use the 'networkx' library in Python.

In the given code snippet, we calculate the average clustering coefficient for the graph, excluding disconnected components:

```
# Calculate the average clustering coefficient for the graph without taken into consideration the disconnected components
avg_clustering = nx.average_clustering(G, count_zeros=False)

# Print the average clustering coefficient
print(f"Average clustering coefficient = {avg_clustering}")
```

```
Average clustering coefficient = 0.4098309494095463
```

The figure below shows the Clustering Coefficient Histogram for all the Actors:



B. Interpretation

The calculated average clustering coefficient of 0.4098 suggests that there is a moderate level of interconnectedness among the nodes in the network. This indicates the presence of some cliques or tightly-knit groups within the network, although not to an extreme degree.

8. COMMUNITY DETECTION

In this section, we will discuss the application of community detection algorithms to identify communities within a network of actors who have acted in movies together. Specifically, we will focus on two methods: the Louvain method and the Infomap algorithm. We will also provide an explanation of the differences between these two approaches in the context of actor collaboration networks.

A. Louvain Method

The Louvain method, is a hierarchical, agglomerative clustering algorithm aimed at maximizing modularity. When applied to an actor collaboration network, the nodes represent actors and the edges represent co-appearances in movies. The weight of an edge between two actors can be determined by the number of movies they have acted in together.

Applying the Louvain method to an actor collaboration network can reveal communities of actors who frequently collaborate or share similar movie genres. The algorithm groups actors into communities that maximize the modularity of the network, which essentially evaluates the density of connections within communities compared to the connections between communities. Time Complexity: O(n log n)

B. Infomap Algorithm

The Infomap algorithm, based on information flow and random walks, can also be applied to actor collaboration networks. In this context, the nodes represent actors, and the edges represent their co-appearances in movies. The weight of an edge between two actors can be determined by the number of movies they have acted in together.

Applying the Infomap algorithm to an actor collaboration network can uncover communities of actors with a strong flow of information, i.e., actors who frequently collaborate or share similar movie genres. The algorithm seeks to minimize the description length of a random walk on the network, which corresponds to the most probable trajectory of a "random walker" moving between actors based on their collaboration history.

C. Method

We first implemented the Infomap algorithm by creating an instance of the infomap.Infomap class. Nodes and edges were added to the algorithm using the add_node and add_link functions, respectively. The algorithm was then executed using the run method. The resulting communities were stored and analyzed.

Next, we utilized the Louvain algorithm provided by the community_louvain library. The algorithm partitions the graph into communities by optimizing the modularity metric. We applied the algorithm to the graph and obtained a partitioning of the nodes into communities.

D. Result

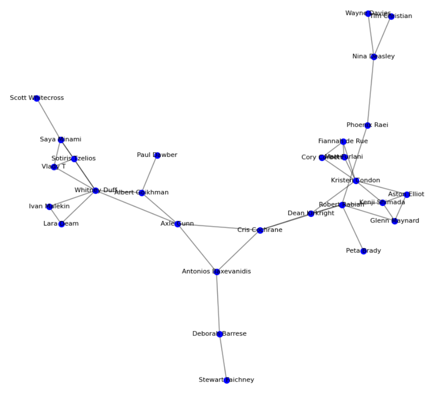
The Infomap algorithm detected a total of 567 communities in the graph, while the Louvain algorithm identified 619 communities. The difference in the number of communities suggests variations in the community structure revealed by each algorithm.

Analyzing the largest community identified by the Infomap algorithm (Community ID: 1), we found it to consist of 17,942 nodes. On the other hand, the largest community detected by the Louvain algorithm exhibited a size of 3,041 nodes (Community ID: 1). The top 20 actors within this largest Louvain community were listed, including Mike Benitez, Chris Coppola, and Lawrence Pressman.

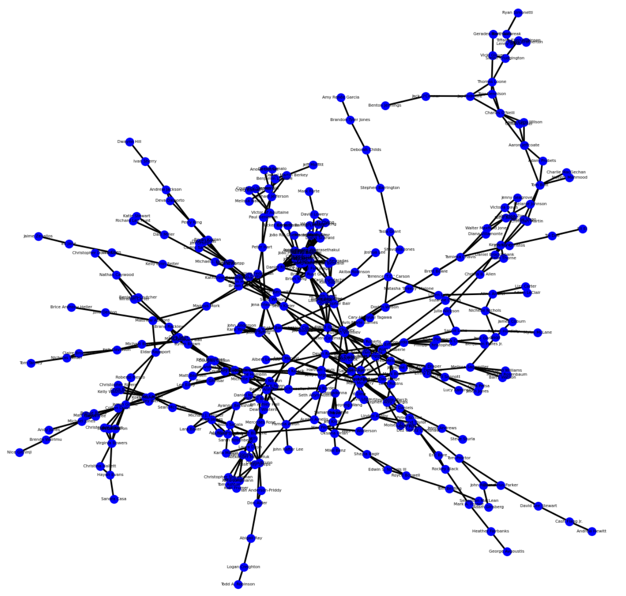
Furthermore, we examined the sizes of the communities detected by both algorithms. The Infomap algorithm identified a community with 1,018 nodes (Community 0) as the largest after the Louvain algorithm. The Louvain algorithm revealed a community with 3,041 nodes (Community 1) as the largest. The distribution of community sizes provides insights into the heterogeneity of the network and the presence of both small and large communities.

E. Visualization

Visualization is an essential aspect of community detection analysis as it provides a visual representation of the detected communities within a network. Here are the visualizations generated for both the Infomap and Louvain algorithms.



In the Infomap visualization, we selected a specific community with the ID 20 to visualize. The nodes belonging to this community were extracted, and a subgraph was created using only these nodes. The spring layout algorithm was used to position the nodes in the visualization. The resulting subgraph was plotted with blue nodes, and customizable parameters such as node size, label size, and edge width were set to enhance the visual clarity. Node labels were drawn separately to improve interpretability.



For the Louvain visualization, we selected a specific community with the ID 29 to visualize. Similar to the Infomap visualization, we extracted the nodes belonging to this community and created a subgraph using these nodes. The spring layout algorithm was used to position the nodes in the visualization. The community subgraph was plotted with blue nodes, and edges were drawn with reduced opacity (alpha=0.5) to enhance clarity. Node labels were added to provide additional information about the nodes within the community.

9. CONCLUSION

In this study, we conducted a network analysis of actors based on their collaborations in movies. Our objective was to gain insights into the structure and dynamics of the film industry, identify closely-knit groups of actors, and find influential actors within the network. We collected a comprehensive dataset of English movies released between 2000 and 2023 using the TMDb API and IMDb web scraping techniques. After data cleaning and preprocessing, we created an actor-network graph where actors were represented as nodes and the connections between them were based on their co-appearances in movies.

Our analysis of the network's properties revealed several key findings. The network exhibited a sparse and modular structure with a low density, indicating that nodes tended to form tight-knit communities with relatively few connections between them. We identified a giant component in the network, suggesting the presence of a small-world property where most nodes could be reached from any other node by a small number of steps. The degree distribution analysis revealed a heterogeneous distribution, with some actors having many more connections than others. This indicated the presence of hubs or highly connected nodes that played a crucial role in information flow and network robustness.

Furthermore, we performed path analysis to find the shortest paths between pairs of actors, providing a measure of the "distance" between them in terms of their professional collaborations. We also computed centrality measures such as degree centrality, betweenness centrality, closeness centrality, and eigenvector centrality to identify the most central and influential actors in the network. Finally, we calculated the clustering coefficient to assess the presence of cliques or tightly-knit groups within the network.

Our study contributes to a deeper understanding of the film industry by uncovering the collaborative dynamics among actors. The insights gained from our analysis can inform industry professionals, researchers, and movie enthusiasts about the structure, trends, and key players within the film industry. Future research can build upon this study by expanding the dataset, considering temporal aspects, and incorporating additional factors such as genre or box office performance to further explore the dynamics of the film industry network.

Overall, network analysis provides a powerful framework for studying complex systems such as the film industry. By examining the relationships between actors, we can uncover hidden patterns, identify influential individuals, and gain a holistic understanding of the collaborative dynamics that shape the film industry.

10. REFERENCES

1. NetworkX 3.1 documentation. (n.d.). *betweennesscentrality*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/genera

2. NetworkX 3.1 documentation. (n.d.). *Clustering*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/cluste

3. Memgraph's Guide for NetworkX library. (n.d.). *Community detection algorithms overview*. Retrieved from https://networkx.guide/algorithms/community-detection/

4. NetworkX 3.1 documentation. (n.d.). *Degree Analysis*. Retrieved from https://networkx.org/documentation/stable/autoexamples/drawing/plo

5. NetworkX 3.1 documentation. (n.d.). *degreecentrality*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/generated/n

6. NetworkX 3.1 documentation. (n.d.). *eigenvectorcentrality*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/generated/n

7. GeeksforGeeks. (2022a). *Betweenness Centrality Centrality Measure*. Retrieved from https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/

8. GeeksforGeeks. (2022b). *Clustering Coefficient in Graph Theory*. Retrieved from https://www.geeksforgeeks.org/clustering-coefficient-graph-theory/

9. GeeksforGeeks. (2023). *How to find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm*. Retrieved from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

10. NetworkX 3.1 documentation. (n.d.). *louvaincommunities*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/generated/n

11. Marsden, P. V. (2005). *Network Analysis*. In *Elsevier eBooks* (pp. 819–825). Retrieved from https://doi.org/10.1016/b0-12-369398-5/00409-6

12. Rita, L. (2021, December 14). *Infomap Algorithm - Towards Data Science*. *Medium*. Retrieved from https://towardsdatascience.com/infomap-algorithm-9b68b7e8b86

13. NetworkX 3.1 documentation. (n.d.). *shortest_path*. Retrieved from https://networkx.org/documentation/stable/reference/algorithms/gener

14. Math Insight. (n.d.). *The degree distribution of a network*. Retrieved from https://mathinsight.org/degreedistribution

15. Topcoder. (n.d.). *Web Scraping with Beautiful Soup*. Retrieved from https://www.topcoder.com/thrive/articles/web-scraping-with-beautiful-soup