SOCIAL NETWORK ANALYSIS

Network Analysis (Round 1)

Datasets Used:

Euroroads
General Relativity and Quantum Cosmology Collaboration Network

Project Report By Team Alpha

Ronit Gupta(21UCS173)
Ruthvik Devavarapu (21UCS174)
Saumya Saraswat(21UCS184)
Saunak Kushwaha(21UCS185)

Course Instructor

Dr. Sakthi Balan

Table Of Contents

Problem Statement

You can choose two datasets (with at least 1000 number of nodes in the network) ONLY from the given links.

Once you fix the datasets, you have to do the following:

- Summarise the network statistics - Plot the degree distribution of the network, find the max and minimum degree, average degree, and Standard deviation of the degree distribution. (Visualisation wherever necessary)
- Find all centrality measures studied in the class - Degree centrality, Eigenvector centrality, Katz and PageRank centrality, clustering coefficients (both local and global), find average local clustering coefficient and compare with global clustering coefficient, find the betweenness and closeness centrality and reciprocity and transitivity that we have studied in the class using appropriate algorithms (you may use specific packages for this or write your algorithm for the same).
- Produce a project report with all details like the:
    - Complete Description of the Data Set - What is the network about - what are nodes and edges depicting in the network?
    - In the dataset, what are the centrality measures you studied (with the top 10 as a table and the visualisation for the complete network)
    - What are the clustering coefficients (local, global) for this data set (with the top 10 as a table and the visualisation for the complete network)
    - Which are all (centrality measures) appropriate for your study?
    - What is the output of all centrality measures? Visualise it graphically. Do not put the value in text format! (no marks if the output is given ONLY in text format).
    - What are your inferences from the output?
    - Compare and contrast the two networks taken according to the centrality measures study and present your inferences from it.

Tools Used

```
#Libraries for Data Manipulation
import numpy as np
import pandas as pd

#Libraries for Graph Plotting
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

#Libraries for Network Analysis
import networkx as nx
```

Besides using some standard Data Manipulation and Visualization packages, we have used **NetworkX,** a Python package, to create, manipulate, and study complex networks' structure, dynamics, and functions to perform the required network analysis in our project.

About NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It provides:

- Tools for the study of the structure and dynamics of social, biological, and infrastructure networks;
- a standard programming interface and graph implementation that is suitable for many applications;
- a rapid development environment for collaborative, multidisciplinary projects;
- an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and
- The ability to work painlessly with large nonstandard data sets.

With NetworkX, you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyse network structure, build network models, design new network algorithms, draw networks, and more.

Network 1 - Euroroads

Overview

This is the international E-road network located primarily in Europe. The network is undirected; nodes represent cities, and an edge between two nodes indicates that an E-road connects them.

- The network we get from the dataset contains nodes representing the cities, and the edges represent the roads between the cities.

Additional Statistics

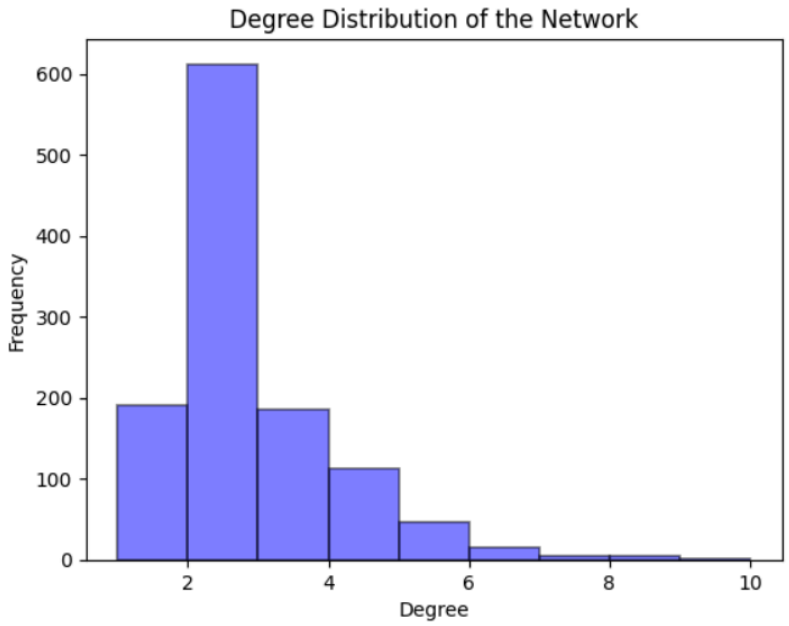
Nodes	1,174
Edges	1,417
Loop count	0
Triangle count	32
Maximum degree	10
Average degree	2.413 97
Fill	0.002 057 94
Size of LCC	1,039
Diameter	62
90-Percentile effective diameter	33.340 5
Mean distance	19.181 2
Power law exponent	2.289 90
Clustering coefficient	0.033 886 3
Algebraic connectivity	0.001 160 30

Network Statistics

Degree Distribution Of This Network

We have used the code below to obtain a degree distribution for our network.

```
degrees = [G.degree(node) for node in G.nodes()]
plt.hist(degrees, bins=range(min(degrees), max(degrees)+1), alpha=0.5, color='b', edgecolor='black', linewidth=1.2)
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the Network')
plt.show()
```



Additional Statistics

The image below presents our code and the obtained results for additional statistics for our chosen network. Besides the number of nodes and edges, we have also found out the following:

- Maximum Degree of a node(city name) in the network.
- Minimum Degree of a node(city name) in the network.
- The Average Degree of a node in the network.
- The Standard deviation of the Degree Distribution in the network.

```python
#Calculating the Degree of Each Node In the Network
degrees = dict(G.degree())

#Find the nodes with maximum and minimum degree
max_degree_node = max(degrees, key=degrees.get)
min_degree_node = min(degrees, key=degrees.get)

#Adjust node numbering and mapping
node_name_mapping = {str(i+1): name for i, name in enumerate(node_names)}  # Adjusted for node numbers
nx.relabel_nodes(G, node_name_mapping, copy=False)

#Print the name of the city with the maximum and minimum degree
print("City with maximum degree:", max_degree_node, "Degree:", degrees[max_degree_node])
print("City with minimum degree:", min_degree_node, "Degree:", degrees[min_degree_node])
```

```
City with maximum degree: Moscow Degree: 10
City with minimum degree: Greenock Degree: 1
```

```python
print('Number of nodes', len(G.nodes))
print('Number of edges', len(G.edges))
avg_degree = np.mean(degrees)
std_dev = np.std(degrees)

print(f"Standard Deviation: {std_dev}")
print(f"Average Degree: {avg_degree}")
```

```
Number of nodes 1174
Number of edges 1417
Standard Deviation: 1.188678457809932
Average Degree: 2.41396933560477
```

Centrality Measures

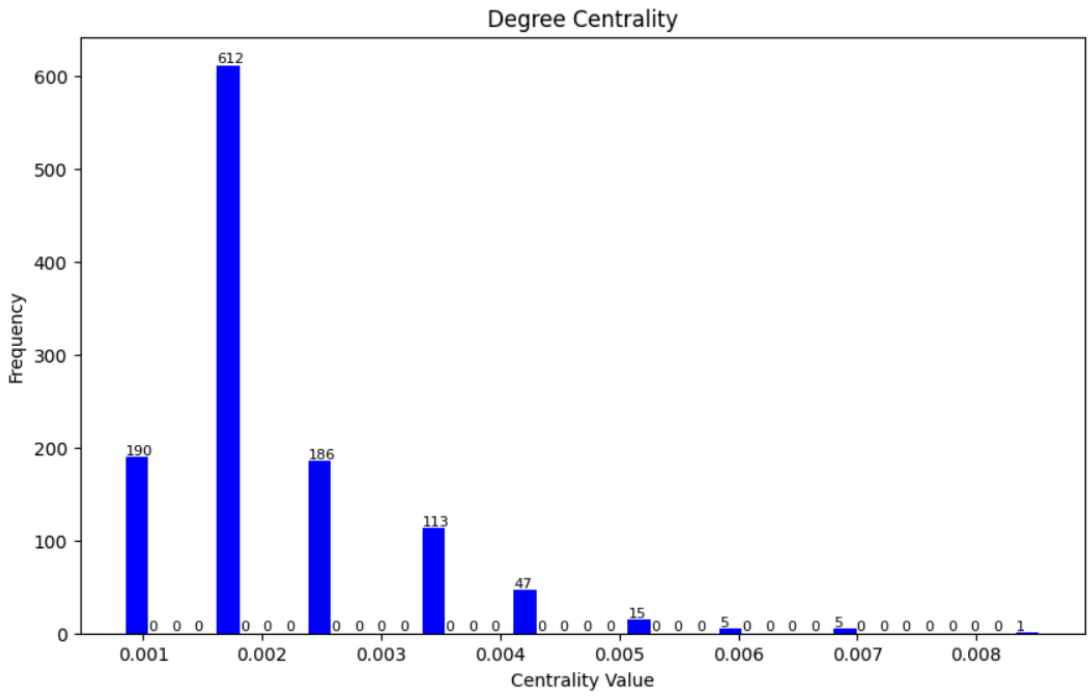
Degree Centrality

Degree centrality in a graph represents the number of edges that are incident upon a node (i.e., the number of connections a node has). In simpler terms, it measures how connected a node is to other nodes in the graph.

In the context of a road network graph, degree centrality would indicate the number of roads that converge at a specific city or intersection. Cities with a high degree of centrality serve as major transportation hubs, where numerous roads intersect, making them pivotal points for travel and connectivity. On the other hand, cities with a low degree of centrality have fewer roads converging at them, indicating a lower level of connectivity and importance in the network.

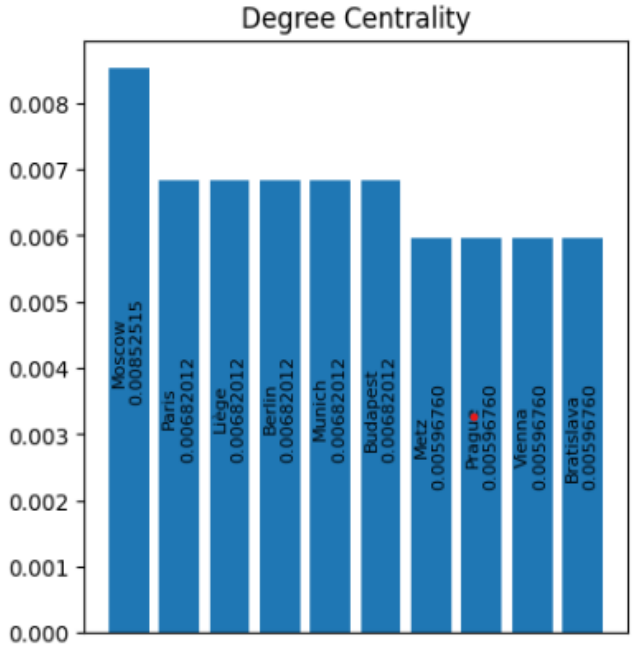
```python
def degree_centrality(G, node):
    neighbors = G[node]
    degree = len(neighbors)
    return degree / (len(G) - 1)

# Calculate degree centrality for each node
deg_centrality = {}
for node in G.nodes():
    deg_centrality[node] = degree_centrality(G, node)

# Print each node and its Degree Centrality
for node, centrality in deg_centrality.items():
    print(f"{node}: Degree Centrality = {centrality}")
```

```python
# Plot distribution of centrality values for Degree Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Degree Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Node	Degree Centrality
Moscow	0.008525
Paris	0.006820
Liege	0.006820
Berlin	0.006820
Munich	0.006820
Budapest	0.006820
Metz	0.005968
Prague	0.005968
Vienna	0.005968
Bratislava	0.005968



Eigenvector Centrality

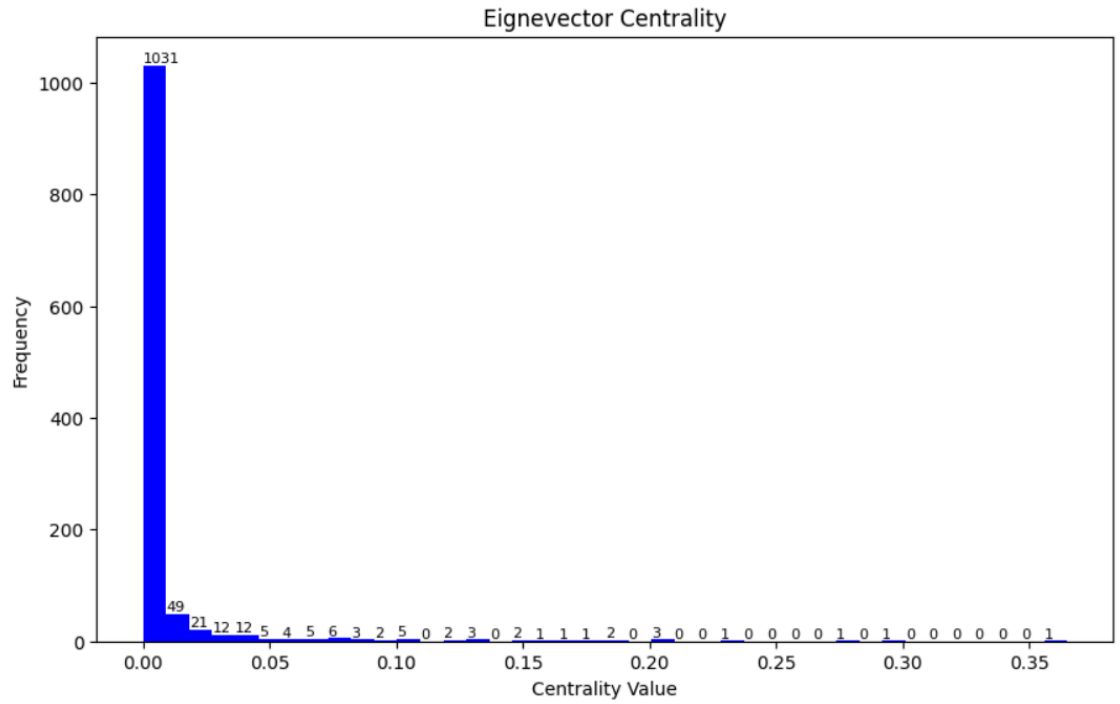
Eigenvector centrality assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question. By calculating the eigenvector centrality of each city, we can identify the most influential cities in the network, which can serve as essential junctions or hubs for transportation and communication between other influential cities.

For example, if a city has a high eigenvector centrality score, it means that it is connected to other cities that also have high centrality scores, indicating that it is a hub for transportation and communication in the network. This information can help plan transportation routes, identify potential bottlenecks in the network, and optimise resource allocation.

Furthermore, eigenvector centrality can also help identify communities or clusters of cities within the network, which can be useful for regional planning and development. By analyzing the eigenvector centrality scores of cities within a region, we can identify key transportation hubs and plan infrastructure development accordingly.

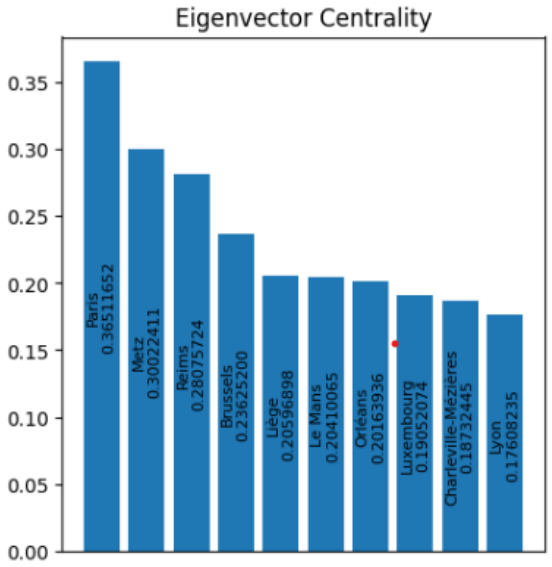
```
#Calculate Eignevector Centrality
eigenvector_centrality = nx.eigenvector_centrality(G)

#Print each node and its Eignevector Centrality
for node, centrality in eigenvector_centrality.items():
    print(f"Node {node}: Eignevector Centrality = {centrality}")
```

```
# Plot distribution of centrality values for Eignevector Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Eignevector Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Nodes	Eigenvector Centrality
Paris	0.365117
Metz	0.300224
Reims	0.280757
Brussels	0.236252
Liege	0.205969
Le Mans	0.204101
Orleans	0.201639
Luxembourg	0.190521
Charleville-Mezieres	0.187324
Lyon	0.176082



Katz Centrality

Katz centrality quantifies the influence of a node based on its immediate neighbours and the neighbours of its neighbours, and so on, with diminishing weights for nodes that are farther away.

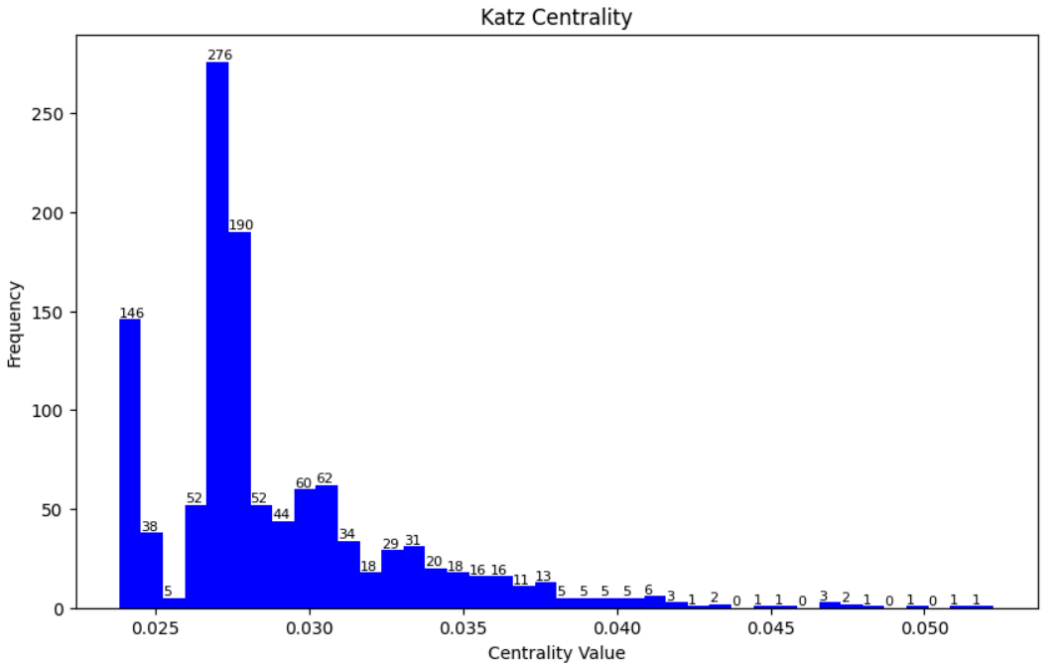
Cities with higher Katz centrality scores are crucial hubs in a transportation network that is well-connected to other cities and linked to well-connected cities. This centrality measure considers direct and indirect connections, favouring cities with strong ties to essential locations. It identifies key bridges and central points, enhancing network efficiency. Analyzing Katz centrality helps pinpoint major transportation hubs, ensuring smoother travel and network accessibility, which is vital for overall road network functionality and efficiency.

```python
def katz(G, alpha=0.1, beta=1.0, normalized=True, weight=None):

    if len(G) == 0:
        return {}
    try:
        nodelist = beta.keys()
        if set(nodelist) != set(G):
            raise nx.NetworkXError("beta dictionary must have a value for every node")
        b = np.array(list(beta.values()), dtype=float)
    except AttributeError:
        nodelist = list(G)
        try:
            b = np.ones((len(nodelist), 1)) * beta
        except (TypeError, ValueError, AttributeError) as err:
            raise nx.NetworkXError("beta must be a number") from err

    A = nx.adjacency_matrix(G, nodelist=nodelist, weight=weight).todense().T
    n = A.shape[0]
    centrality = np.linalg.solve(np.eye(n, n) - (alpha * A), b).squeeze()

    # Normalize: rely on truediv to cast to float, then tolist to make Python numbers
    norm = np.sign(sum(centrality)) * np.linalg.norm(centrality) if normalized else 1
    return dict(zip(nodelist, (centrality / norm).tolist()))

katz_centrality = katz(G, alpha=0.1, beta=1.0, normalized=True, weight=None)

# Print each node and its Katz Centrality
for node, centrality in katz_centrality.items():
    print(f"Node {node}: Katz Centrality = {centrality}")
```

Parameter Description

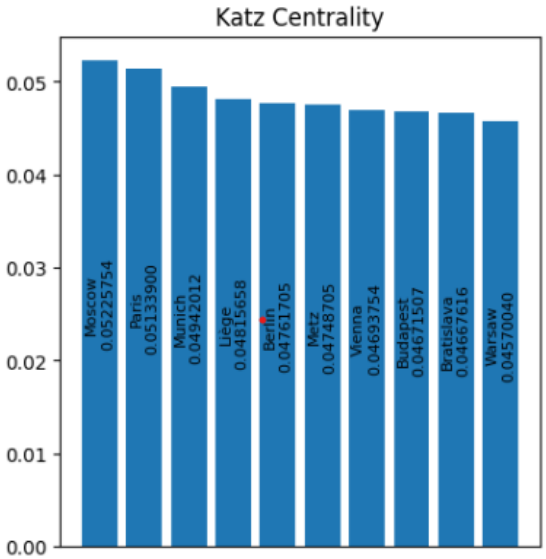
1. **alpha**: Attenuation factor for the Katz centrality computation. It controls the impact of indirect paths on centrality. A smaller alpha reduces the influence of distant nodes, while a more significant alpha gives more weight to longer paths.
2. **beta**: A parameter that allows you to add a constant to each node's centrality, which can help control centrality values. It can help adjust centrality scores to fit specific requirements or to balance the influence of different factors.
3. **normalized**: If True, the Katz centrality scores are normalized to sum up to 1. Normalization can be helpful when comparing centrality scores across different networks or when interpreting the scores in a relative manner.
4. **weight**: None, string, or dictionary. If None, all edge weights are considered equal (1). If a string, this should be the key for edge weight in the edge data dictionary. If a dictionary, keys are node pairs and values are the edge weight between nodes.

```python
# Plot distribution of centrality values for Katz Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Katz Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Node	Katz Centrality
Moscow	0.052258
Paris	0.051339
Munich	0.049420
Liege	0.048157
Berlin	0.047617
Metz	0.047487
Vienna	0.046938
Budapest	0.046715
Bratislava	0.046676
Warsaw	0.045700



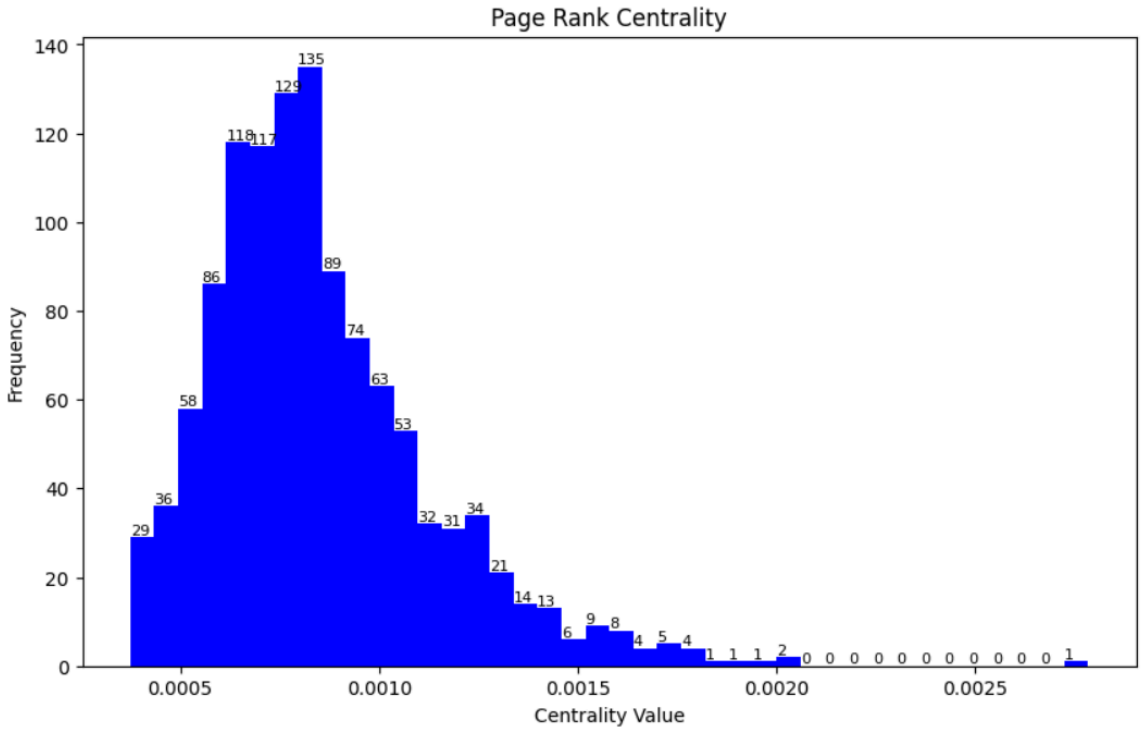
PageRank Centrality

The PageRank of a network measures the importance or centrality of a node within the network. It is calculated based on the number and quality of connections between nodes.

The PageRank measure helps understand a transportation network's overall structure and functionality. It can help identify key transportation hubs and bottlenecks and potential areas for improvement in connectivity and accessibility. By understanding the relative importance of different cities in the network, transportation planners can make more informed decisions about where to allocate resources and how to improve the overall efficiency of the transportation system.

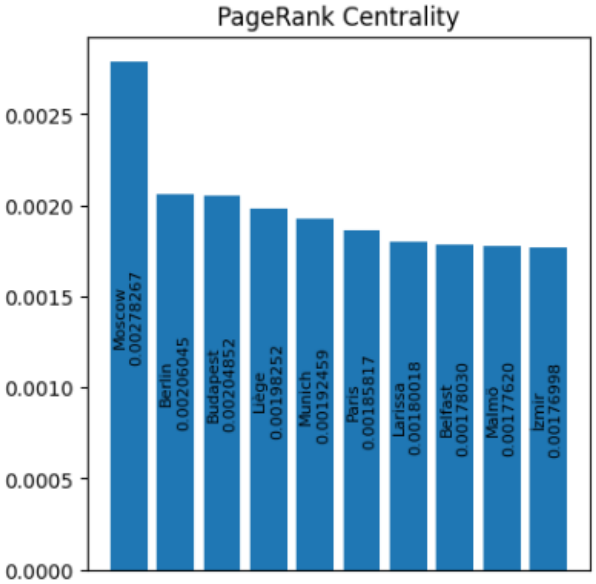
```
# Calculate Page Rank

#alpha: Damping parameter for PageRank, representing the probability of following a link on the graph vs. jumping to a random node.
#       A higher alpha means more weight is given to following links, while a lower alpha gives more weight to jumping to random nodes.
#       The default value of 0.8 is commonly used and provides a good balance between these two factors.

def page_rank(G, alpha=0.85, max_iter=100, tol=1.0e-6, nstart=None, weight='weight'):

    if G.is_directed():
        in_degree = dict(nx.in_degree(G, weight=weight))
        min_in_degree = min(in_degree.values())
        if min_in_degree == 0:
            raise nx.NetworkXError("Graph contains nodes with no incoming edges.")
    else:
        in_degree = dict(nx.degree(G, weight=weight))

    if nstart is None:
        r = {node: 1/len(G) for node in G}
    else:
        r = nstart

    for _ in range(max_iter):
        rlast = r.copy()
        # Power iteration
        for n in G:
            r[n] = (1 - alpha) / len(G) + alpha * sum(rlast[m] / in_degree[m] for m in G.neighbors(n) if m in rlast)
        # check for convergence
        err = max(abs(r[n] - rlast[n]) for n in r)
        if err < tol:
            return r

    return r

# Calculate Page Rank for each node
pr = page_rank(G, alpha=0.8)

# Print each node and its Page Rank
for node, centrality in pr.items():
    print(f"Node {node}: Page Rank = {centrality}")
```

```
# Plot distribution of centrality values for Page Rank Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Page Rank Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Node	Page Rank Centrality
Moscow	0.002783
Berlin	0.002060
Budapest	0.002049
Liege	0.001983
Munich	0.001925
Paris	0.001858
Larissa	0.001800
Belfast	0.001780
Malmo	0.001776
Izmir	0.001770



Betweenness Centrality

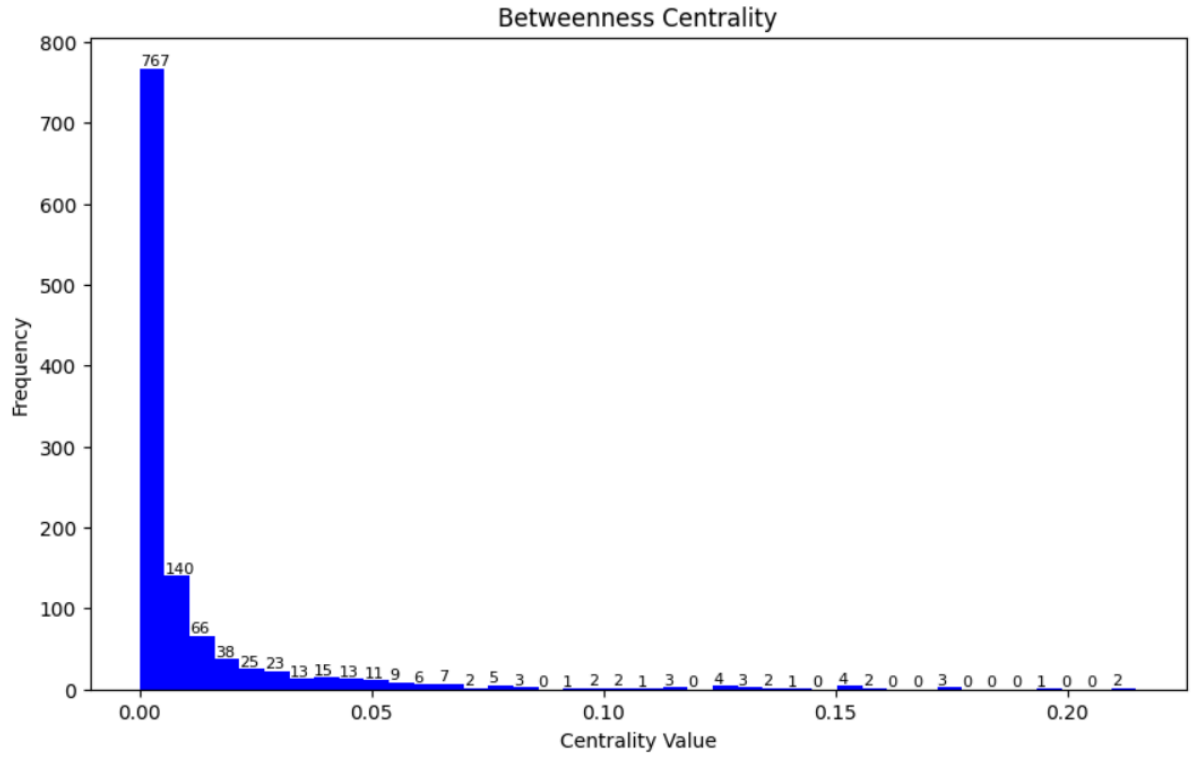
Betweenness centrality in a graph represents the extent to which a node lies on the shortest paths between other nodes. Specifically, it measures the number of times a node acts as a bridge along the shortest path between pairs of other nodes in the graph.

In the context of a road network represented as a graph, betweenness centrality is a measure that identifies the importance of a node based on the number of shortest paths that pass through it. Nodes with high betweenness centrality act as critical connectors within the network, as they lie on many of the shortest paths between pairs of nodes. These nodes facilitate efficient traffic flow and connectivity within the road network.

Identifying nodes with high betweenness centrality is crucial for various transportation planning and management aspects. Firstly, these nodes are key locations for transportation infrastructure planning because they represent strategic points where significant traffic converges. By focusing on these nodes, urban planners and engineers can prioritize investments in infrastructure development to enhance connectivity and alleviate congestion in the network.

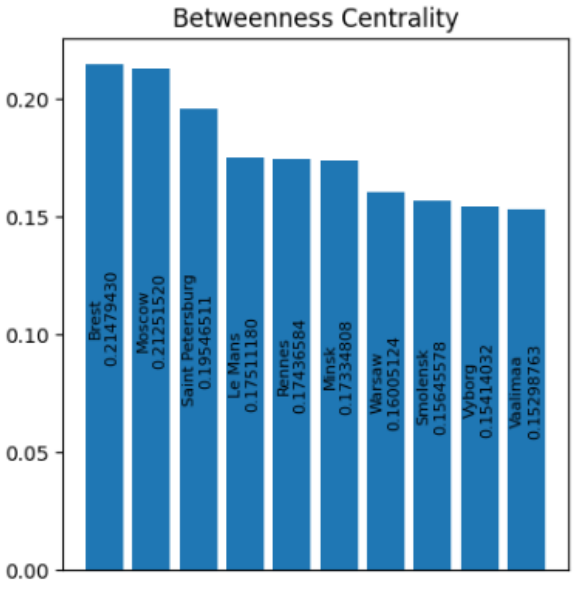
```python
#Calculate Betweenness Centrality

#normalized: If True, the betweenness centrality scores are normalized by dividing by the maximum possible betweenness centrality.
#            Normalization is useful for comparing centrality scores across graphs of different sizes.

#endpoints: If False, only considers paths between nodes as they are internal to the node set.
#           If True, considers both endpoints of the paths as adjacent to other nodes outside the node set.
#           Setting endpoints to False is often appropriate when analyzing networks where nodes represent entities that cannot pass through each other.

bet_centrality = nx.betweenness_centrality(G, normalized = True,endpoints = False)

#Print each node and its Betweenness Centrality (excluding nodes with centrality = 0)
for node, centrality in bet_centrality.items():
    if centrality != 0:
        print(f"Node {node}: Betweenness Centrality = {centrality}")
```

```python
# Plot distribution of centrality values for Betweenness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Betweenness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Nodes	Betweenness Centrality
Brest	0.214794
Moscow	0.212515
Saint Petersburg	0.195465
Le Mans	0.175112
Rennes	0.174366
Minsk	0.173348
Warsaw	0.160051
Smolensk	0.156456
Vyborg	0.154140
Vaalimaa	0.152988



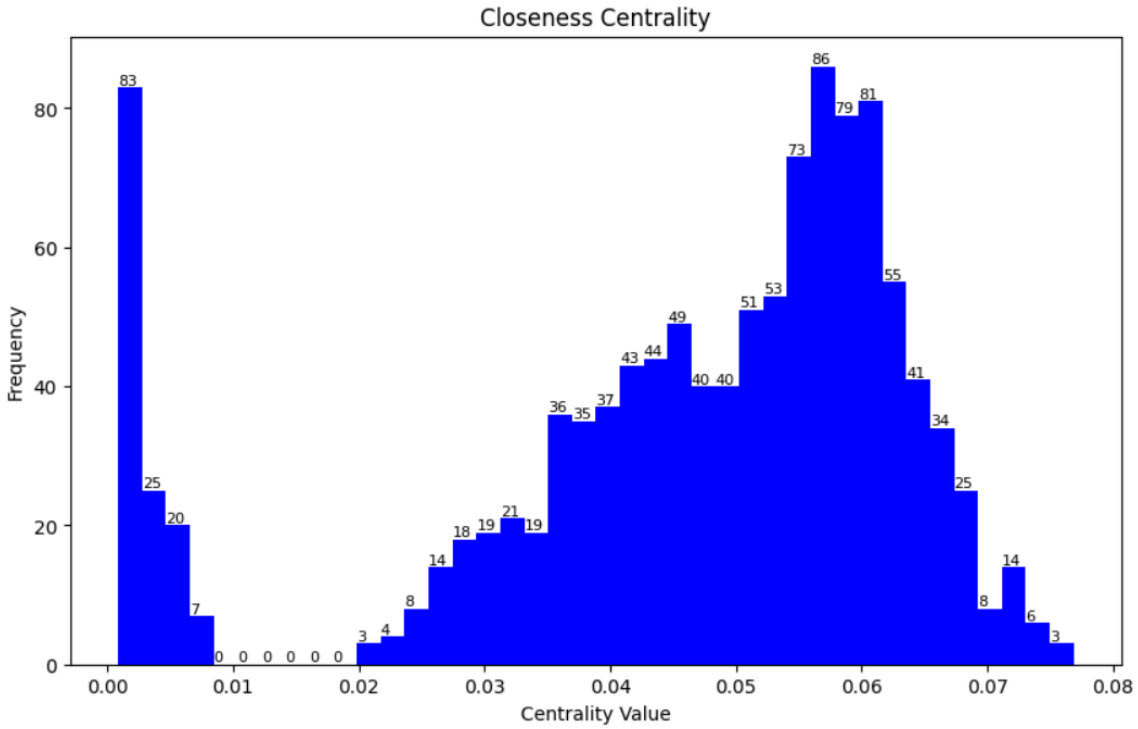
Closeness Centrality

Closeness centrality in a graph represents how close a node is to all other nodes. Specifically, it measures the average shortest path length from a node to all other nodes in the graph.

In a road network, cities with higher closeness centrality are typically more centrally located and have shorter average distances to reach other cities. This means they are potentially more important or influential regarding transportation and connectivity. For example, a city with a high closeness centrality score could quickly reach other cities in the network, making it an ideal location for transportation hubs or distribution centres.

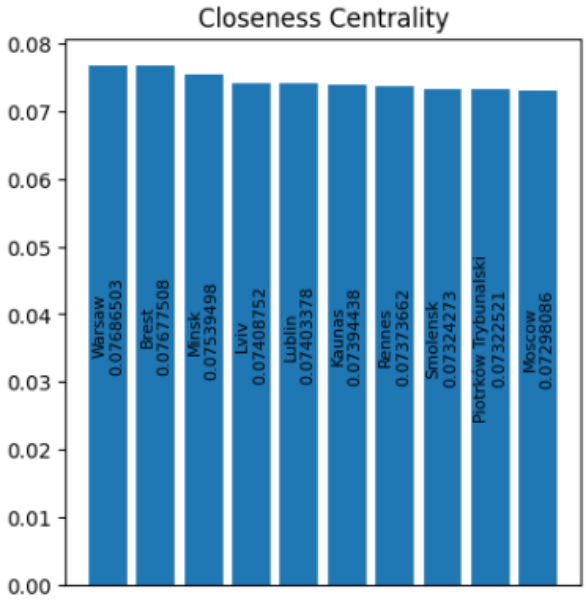
```
#Calculate Closeness centrality
close_centrality = nx.closeness_centrality(G)

#Print each node and its Closeness centrality
for node, centrality in close_centrality.items():
    print(f"Node {node}: Closeness centrality = {centrality}")
```

```
# Plot distribution of centrality values for Closeness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Closeness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

Nodes	Closeness Centrality
Warsaw	0.076865
Brest	0.076775
Minsk	0.075395
Lviv	0.074088
Lublin	0.074034
Kaunas	0.073944
Rennes	0.073737
Smolensk	0.073243
Piotrkow Trybunalski	0.073225
Moscow	0.072981

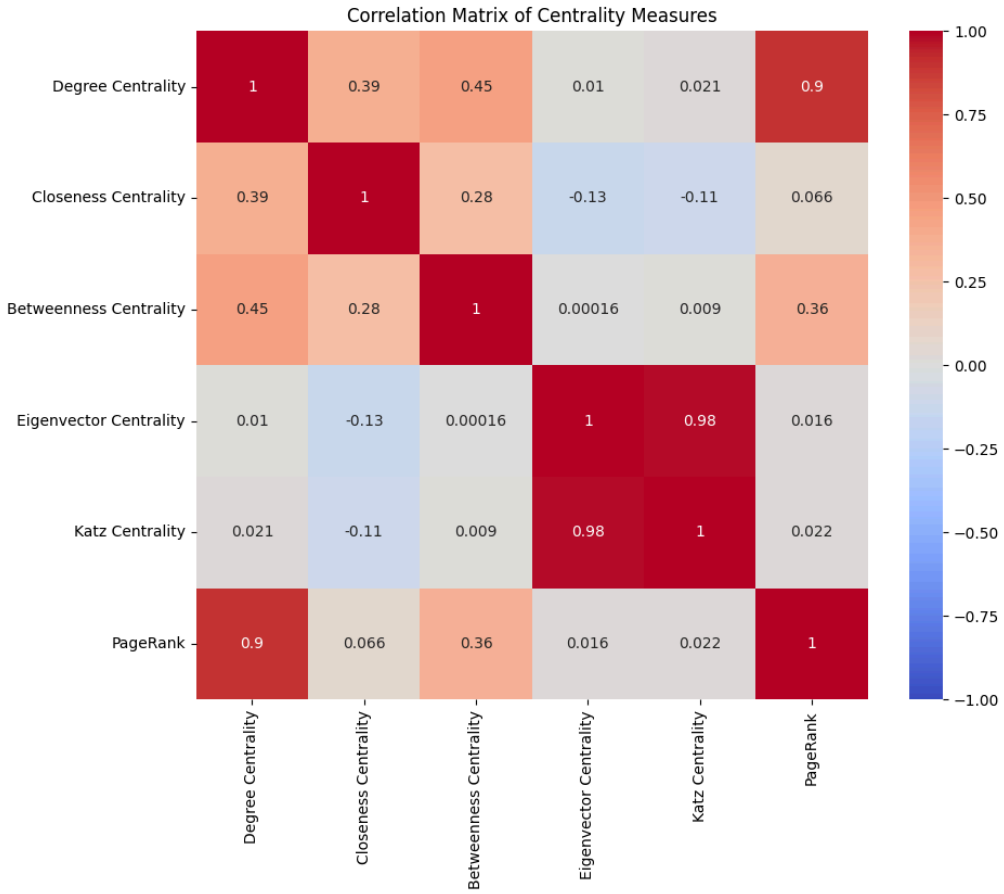


Correlation Matrix Between The Centrality Measures

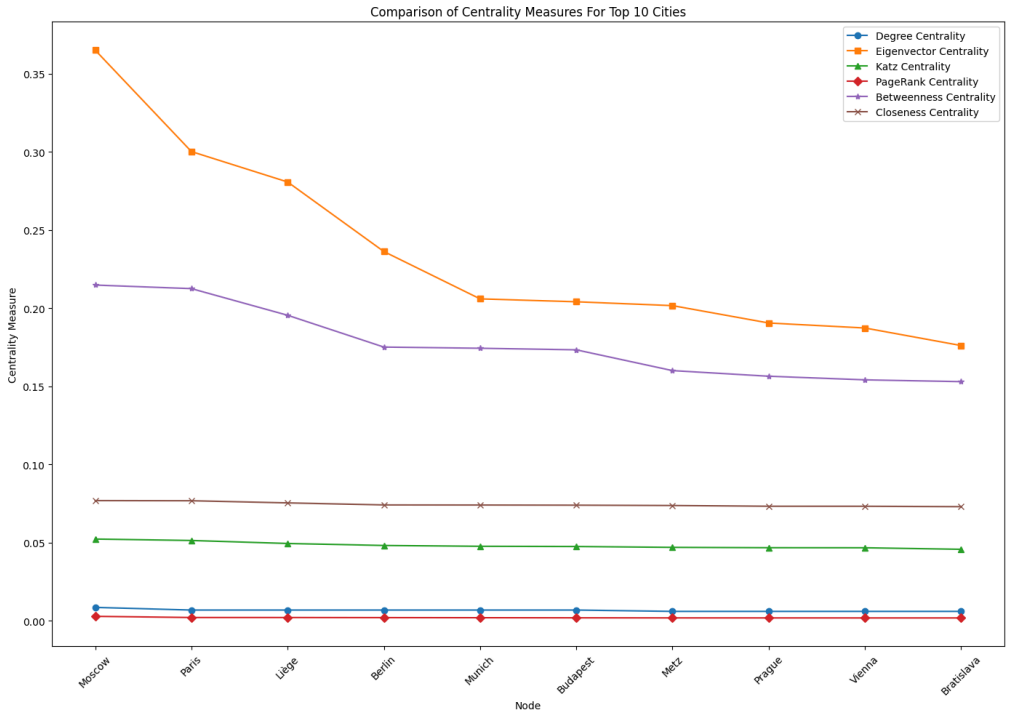
The correlation matrix in the search results shows the correlation between different centrality measures for the European road network. A high correlation between two variables can be beneficial as it indicates a strong relationship. This can simplify analysis and decision-making processes in many cases, as changes in one variable will likely be reflected in another. For example, in network analysis, a high correlation between centrality measures like eigenvector and Katz centrality can provide more confidence in identifying key nodes that play important roles in the network. This can help target resources or interventions more effectively, as nodes with high centrality in one measure are likely to have high centrality in the other, indicating their critical role in the network's structure and function.

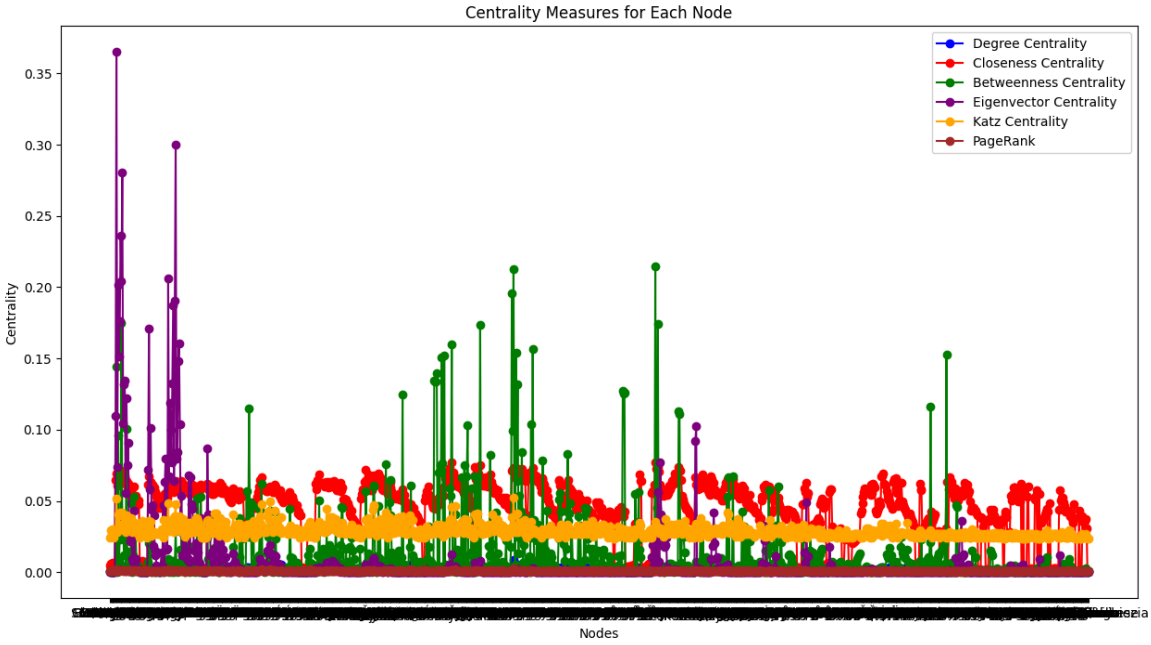
The correlation between degree and PageRank is 0.9, indicating a strong positive correlation between these measures. This means that cities with a high degree of centrality also tend to have a high PageRank centrality, indicating that they are well-connected and linked to other important cities.

However, the correlation between closeness and PageRank is only 0.067, indicating a weak positive correlation between these two measures. This means that cities with a high closeness centrality do not necessarily have a high PageRank centrality, indicating that these two measures capture different aspects of network centrality.



Visualizing Centrality Measures





Clustering Coefficients

Global Clustering Coefficient

```
#Calculate Global Clustering Coefficient
global_clustering_coefficient = nx.transitivity(G)

#Print Global Clustering Coefficient
print("Global Clustering Coefficient:", global_clustering_coefficient)
```

```
Global Clustering Coefficient: 0.6298424741263426
```

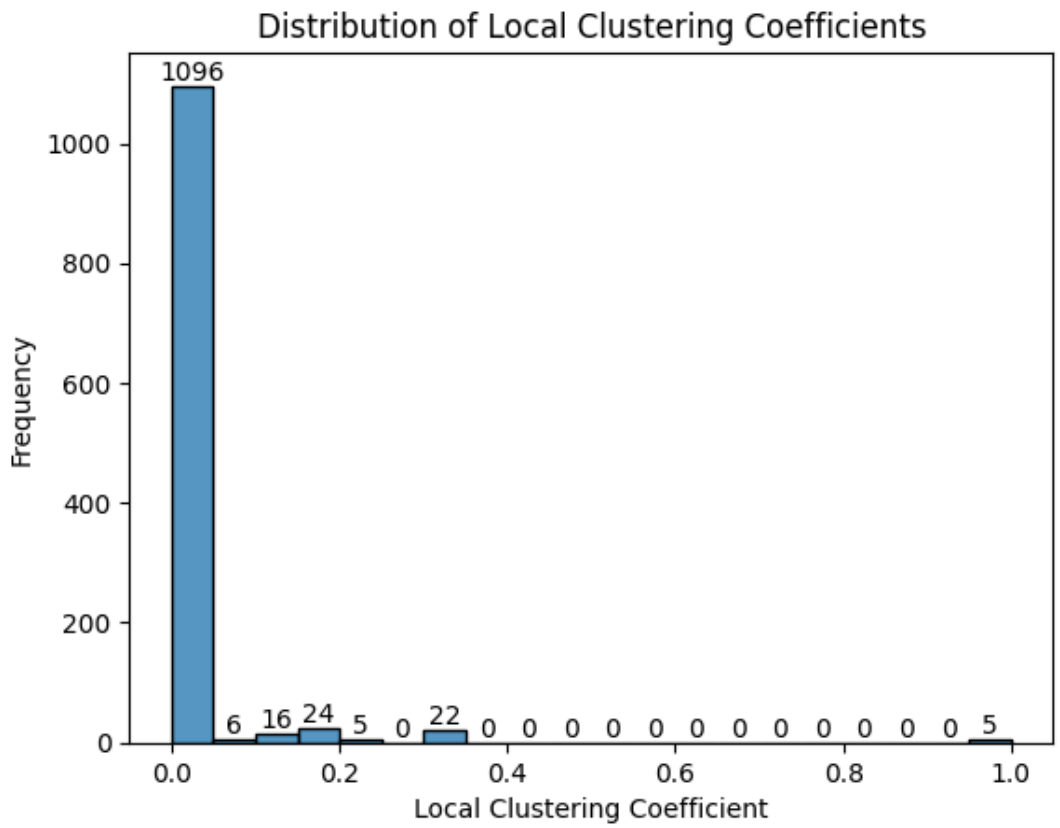
Local Clustering Coefficient

```
#Calculate Local Clustering Coefficient
local_clustering_coefficient = nx.clustering(G)

#Print each node and it's Clustering Coefficient
for node, local in local_clustering_coefficient.items():
    print(f"Node {node}: Local Clustering Coefficient = {local}")
```

```python
#Create a table for the top 10 nodes with highest local clustering coefficients
top_10_nodes = sorted(local_clustering_coefficient.items(), key=lambda x: x[1], reverse=True)[:10]
print("Top 10 Nodes with Highest Local Clustering Coefficients:")
print("Node\tLocal Clustering Coefficient")
for node, coeff in top_10_nodes:
    print(f"{node}\t{coeff}")
```

```
Top 10 Nodes with Highest Local Clustering Coefficients:
Node    Local Clustering Coefficient
Brindisi        1.0
Domokos 1.0
Vyšné Nemecké   1.0
Postojna        1.0
Spielfeld       1.0
Lausanne        0.3333333333333333
Tortona 0.3333333333333333
Brescia 0.3333333333333333
Ulm     0.3333333333333333
Naples  0.3333333333333333
```

```python
#Create a line chart for the distribution of local clustering coefficients
coefficients = list(local_clustering_coefficient.values())
coefficients.sort()
unique_coefficients = list(set(coefficients))
counts = [coefficients.count(coeff) for coeff in unique_coefficients]

#Create a histogram for the distribution of local clustering coefficients
sns.histplot(list(local_clustering_coefficient.values()), bins=20, kde=False)
plt.xlabel('Local Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('Distribution of Local Clustering Coefficients')

#Display the frequency count on top of each bin
for p in plt.gca().patches:
    plt.gca().annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.get_height()),ha='center', va='bottom', color='black', fontsize=10)

plt.show()
```



Many nodes have low local clustering coefficients, which suggests that these cities have relatively few connections to their neighbouring cities. This could indicate that these cities are less central in the European road network and may not play as significant a role in transportation and connectivity as other cities with higher local clustering coefficients. However, it is important to note that a low local clustering coefficient does not necessarily mean a city is unimportant in the network. Other centrality measures, such as degree centrality or Katz centrality, may still indicate that the city is well-connected to other cities in the network, even if it is not well-connected to its immediate neighbours.

Average Local Clustering Coefficient

```
#Calculate Average Local Clustering Coefficient
average_local_clustering_coefficient = nx.average_clustering(G)

print("Average Local Clustering Coefficient:", average_local_clustering_coefficient)
```

```
Average Local Clustering Coefficient: 0.529635811052136
```

Transitivity and Reciprocity Values

Transitivity

```
transitivity = nx.transitivity(G)
print("Transitivity:", transitivity)
```

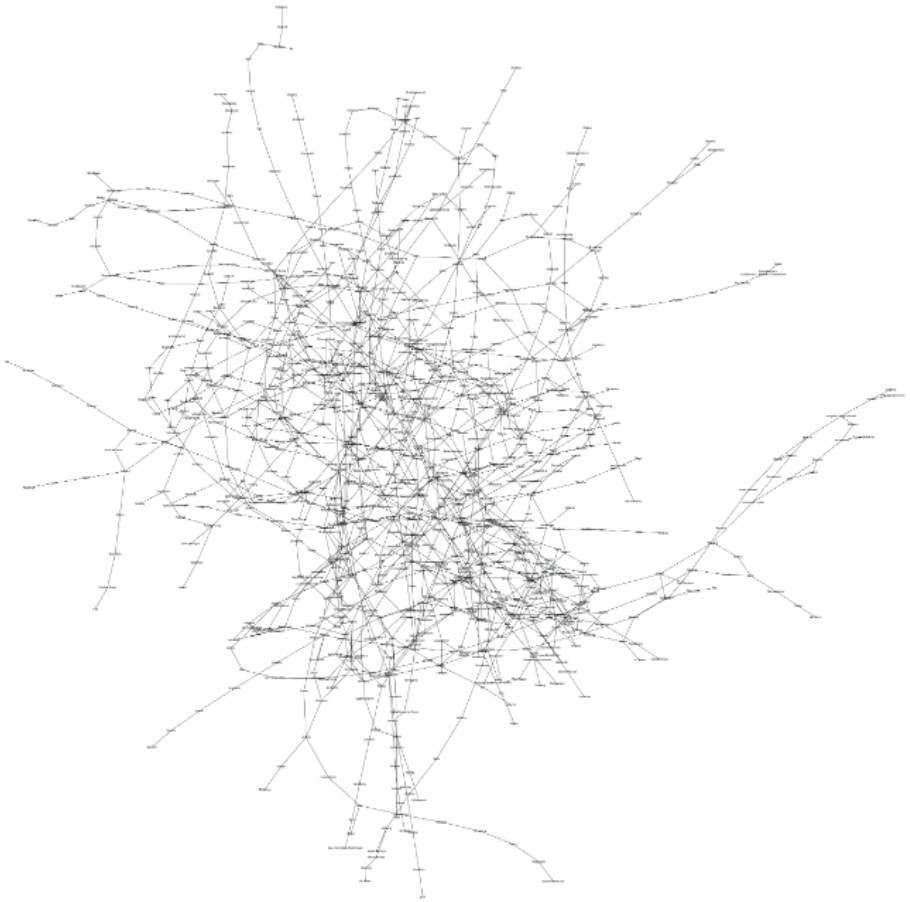
```
Transitivity: 0.0338863395693611
```

Reciprocity

Reciprocity is a simplified version of transitivity because it considers closed loops of length 2, which can only happen in directed graphs. However, our chosen network, in this case, is undirected. Hence, we cannot find a value for the reciprocity of this network.

Largest Connected Component

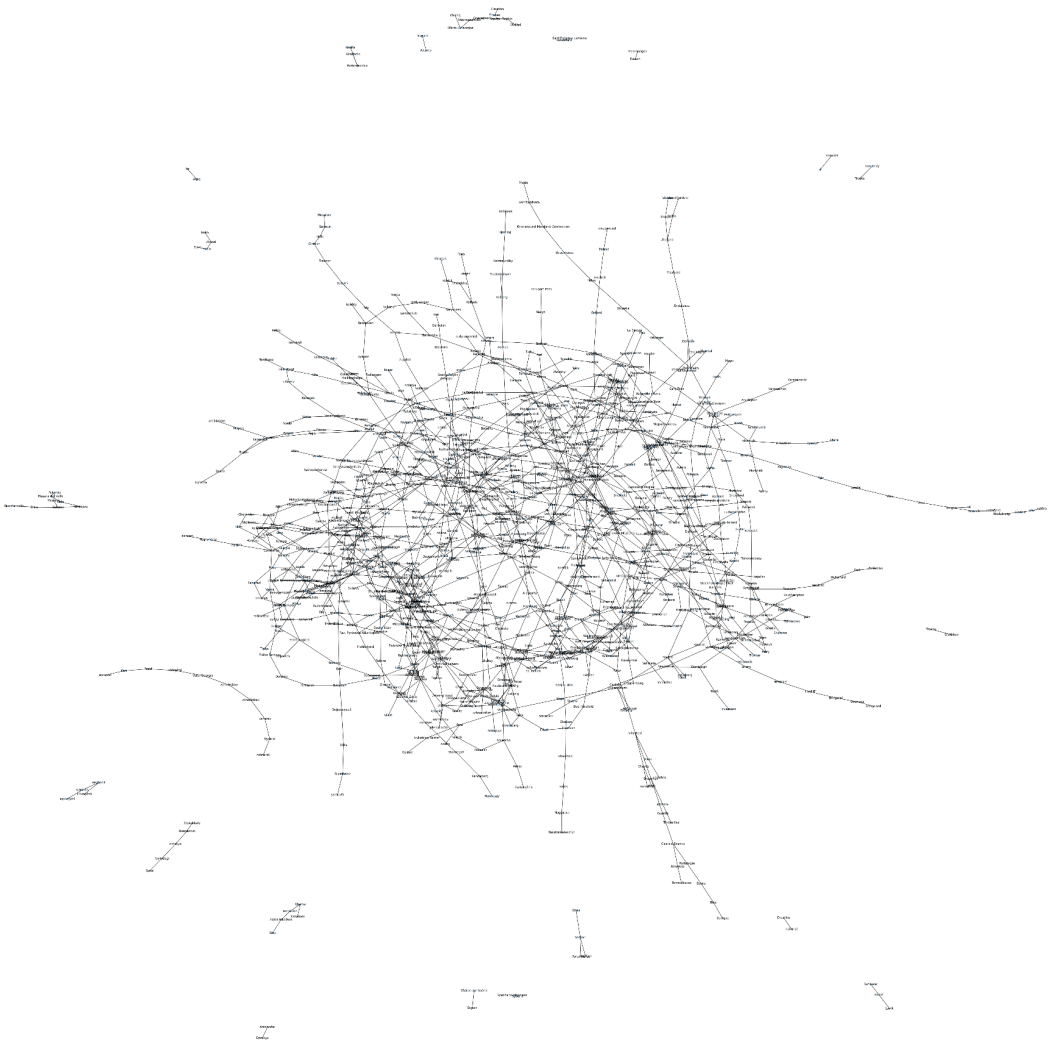
```
#Find the largest connected component
largest_component = max(nx.connected_components(G), key=len)

#Create a subgraph of the largest connected component
largest_component_subgraph = G.subgraph(largest_component)

#Print the number of nodes and edges in the largest connected component
print("Number of nodes in the largest connected component:", largest_component_subgraph.number_of_nodes())
print("Number of edges in the largest connected component:", largest_component_subgraph.number_of_edges())

#Visualize the largest connected component
plt.figure(figsize=(18, 12))
nx.draw(largest_component_subgraph, with_labels=False, node_size = 10)
plt.title('Largest Connected Component')
plt.show()
```



```
Number of nodes in the largest connected component: 1039
Number of edges in the largest connected component: 1305
```

Complete Network Visualization

```
plt.figure(figsize=(56, 56))
nx.draw(G, with_labels=True, node_size = 10, font_size = 10)
```



Network 2 - General Relativity and Quantum Cosmology Collaboration Network

This dataset covers the scientific collaborations between authors' papers submitted to the General Relativity and Quantum Cosmology categories. The network we get from the dataset we chose contains nodes representing the authors of the paper and the edges representing the collaborations between the authors.

Overview

In this network, there is an undirected edge from an author to the co-author of a paper.

- If multiple authors write a paper, then the graph between them is a complete subgraph.
    - If there are *n* authors, then the number of nodes in the subgraph will be *n,* and the total number of edges in this subgraph will be *n!*

This particular dataset covers papers published between January 1993 and April 2003.

Additional Statistics

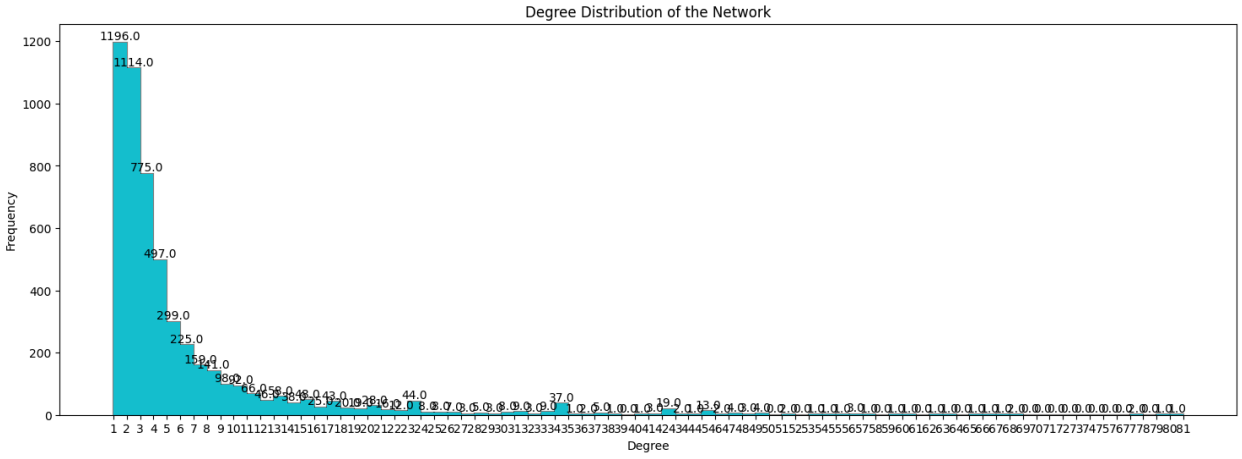
Nodes	5242
Edges	14496
Nodes in the largest WCC	4158(0.793)
Edges in the largest WCC	13428(0.926)
Nodes in the largest SCC	4158(0.793)
Edges in the largest SCC	13428(0.926)
Average clustering coefficient	0.5296
Number of triangles	48260
Fraction of closed triangles	0.3619
Diameter(longest shortest path)	17
90-percentile effective diameter	7.6

Network Statistics

Degree Distribution Of This Network

We have used the code below to obtain a degree distribution for our network.

```python
#Define the Bin Size and Bins
bin_size = 1
bins = range(min(degrees), max(degrees) + bin_size, bin_size)

#Create Histogram
plt.figure(figsize=(18, 6))
plt.hist(degrees, bins=bins, alpha=0.5, color='blue', edgecolor='black', linewidth=1.5)

#Add Text Labels for Bin
for i, bin_value in enumerate(bins[:-1]):
    plt.text(bin_value + bin_size/2, plt.hist(degrees, bins=bins)[0][i], str(plt.hist(degrees, bins=bins)[0][i]), ha='center', va='bottom')

#Set x-axis tick locations and labels
plt.xticks(range(min(degrees), max(degrees)+1, 1))

#Set labels and title
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the Network')

#Show plot
plt.show()
```



Additional Statistics

The image below presents our code and the obtained results for additional statistics for our chosen network. Besides the number of nodes and edges, we have also found out the following:

- Maximum Degree of a node in the network.
- Minimum Degree of a node in the network.
- The Average Degree of a node in the network.
- The Standard deviation of the Degree Distribution in the network.

```
print('Nodes:', len(G.nodes))
print('Edges:', len(G.edges))

max_degree = max(degrees)
min_degree = min(degrees)
avg_degree = np.mean(degrees)
std_degree = np.std(degrees)

print("Max Degree of a Node:", max_degree)
print("Min Degree of a Node:", min_degree)
print("Average Degree of Nodes:", avg_degree)
print("Standard Deviation of Degree Distribution:", std_degree)
```

```
Nodes: 5242
Edges: 14496
Max Degree of a Node: 81
Min Degree of a Node: 1
Average Degree of Nodes: 5.530713468141931
Standard Deviation of Degree Distribution: 7.918407301678446
```

Centrality Measures

Degree Centrality

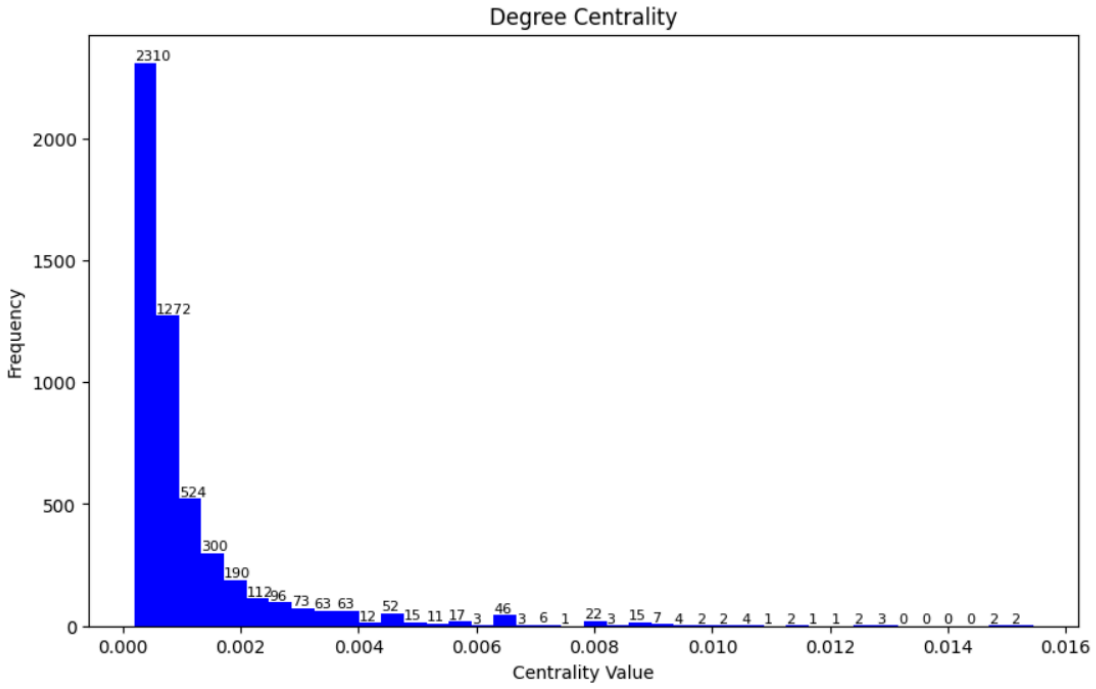
This graph describes the authors and their number of collaborations. The value of the y-axis extends up to 1, starting from 0. The value on the y-axis is very small, which suggests that the number of collaborations is less.

The graph shows that no one has collaborated with all the authors because no node shows the centrality measure 1.

The network size can influence the Degree of centrality because an author who has collaborated with 10 authors in a small network may have higher centrality than an author who has collaborated with 20 authors in a large network.

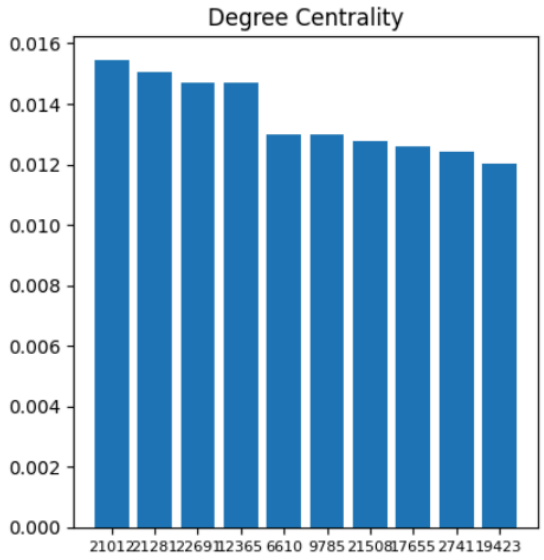
```
#Calculate Degree Centrality
deg_centrality = nx.degree_centrality(G)

#Print each node and its Degree Centrality
for node, centrality in deg_centrality.items():
    print(f"Node {node}: Degree Centrality = {centrality}")
```

```python
# Plot distribution of centrality values for Degree Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Degree Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	Degree Centrality
21012	0.015455
21281	0.015073
22691	0.014692
12365	0.014692
6610	0.012975
9785	0.012975
21508	0.012784
17655	0.012593
2741	0.012402
19423	0.012021



Eigenvector Centrality

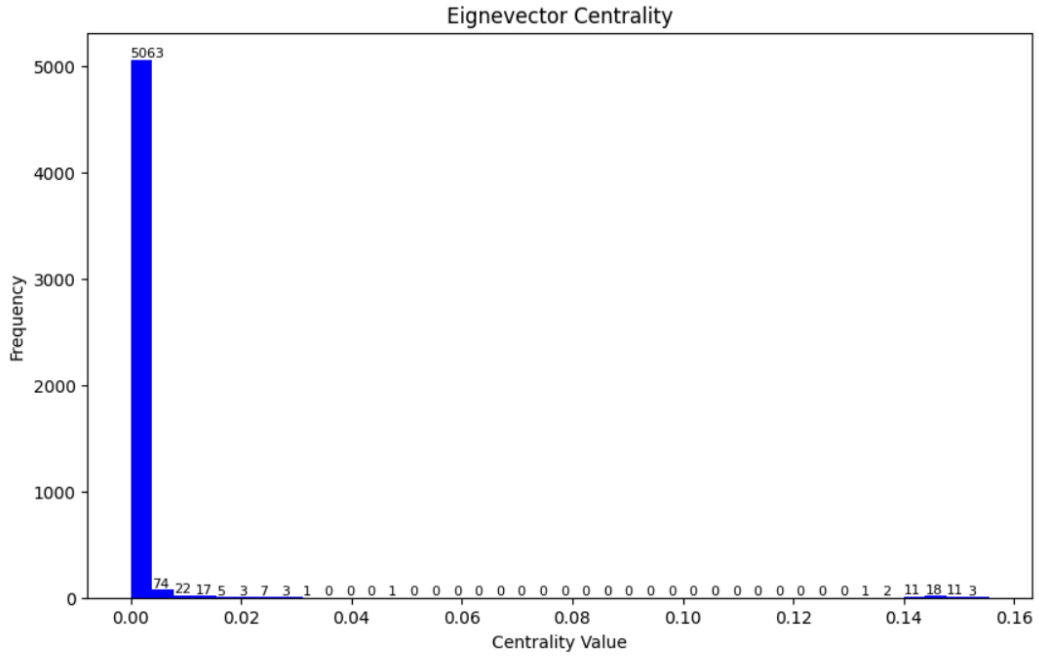
Some authors have a high centrality value, representing that they are more influential in research. It is difficult to conclude who the most influential person is from the graph alone, but the nodes with the highest eigenvector centrality could be the most influential authors.

However, using this centrality measure has a few drawbacks, mentioned below:

- This graph won't be able to explain how strong or important the collaboration between 2 authors is for a particular paper.
- Eigenvector centrality is a relative measure because the centrality of a particular node depends on the centrality scores of other authors with whom they collaborate.

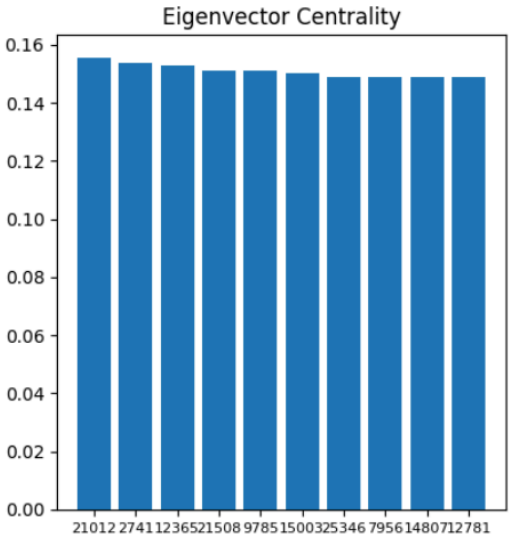
```
#Calculate Eignevector Centrality
eigenvector_centrality = nx.eigenvector_centrality(G)

#Print each node and its Eignevector Centrality
for node, centrality in eigenvector_centrality.items():
    print(f"Node {node}: Eignevector Centrality = {centrality}")
```

```
# Plot distribution of centrality values for Eignevector Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Eignevector Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	EigenVector Centrality
21012	0.155561
2741	0.153574
12365	0.153072
21508	0.151194
9785	0.150903
15003	0.150407
25346	0.149069
7956	0.149066
14807	0.149007
12781	0.148881

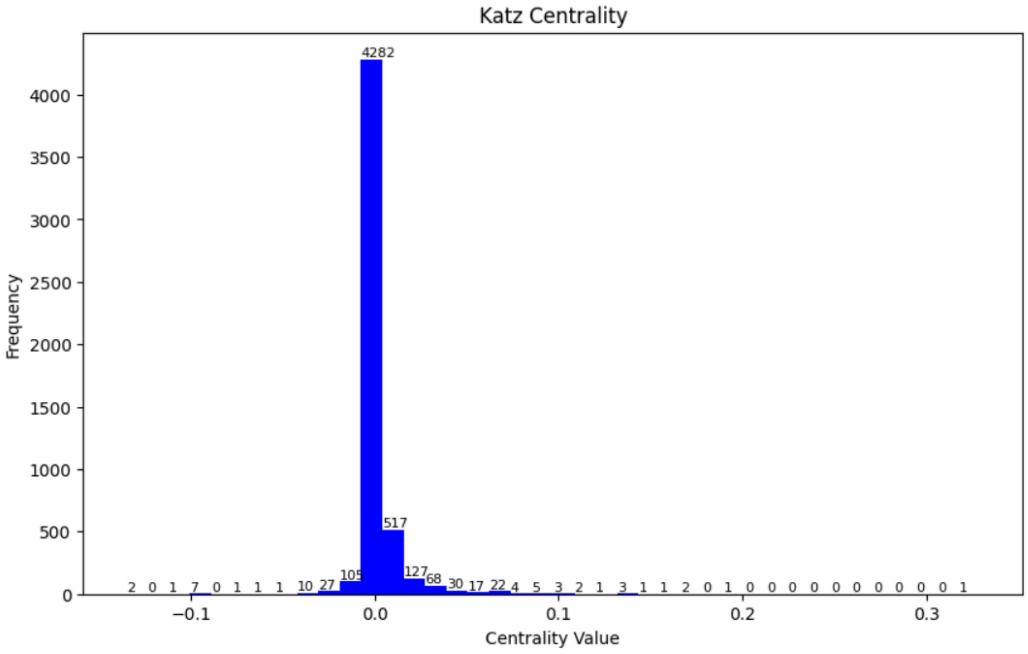


Katz Centrality

Katz centrality generally infers the potential influence of a node in the network based on the length of the paths connecting to other nodes. An author with many short paths leading to them is likely to be a greater influence when compared to one with fewer or long paths. Katz Centrality can also be used to identify the key players in spreading information or ideas through the collaboration network

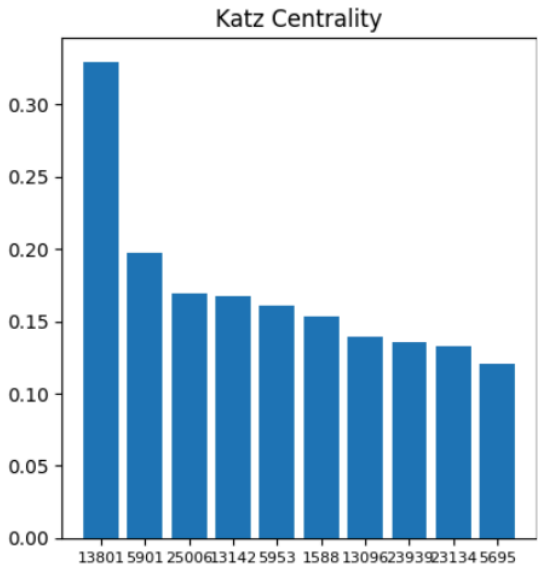
```
katz_centrality = nx.katz_centrality_numpy(G, alpha=0.1, beta=1.0, normalized=True, weight=None)

# Print each node and its Katz Centrality
for node, centrality in katz_centrality.items():
    print(f"Node {node}: Katz Centrality = {centrality}")
```

```
# Plot distribution of centrality values for Katz Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Katz Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	Katz Centrality
13801	0.329264
5901	0.196795
25006	0.168997
13142	0.167177
5953	0.161180
1588	0.153601
13096	0.139432
23939	0.135106
23134	0.132340
5695	0.120516

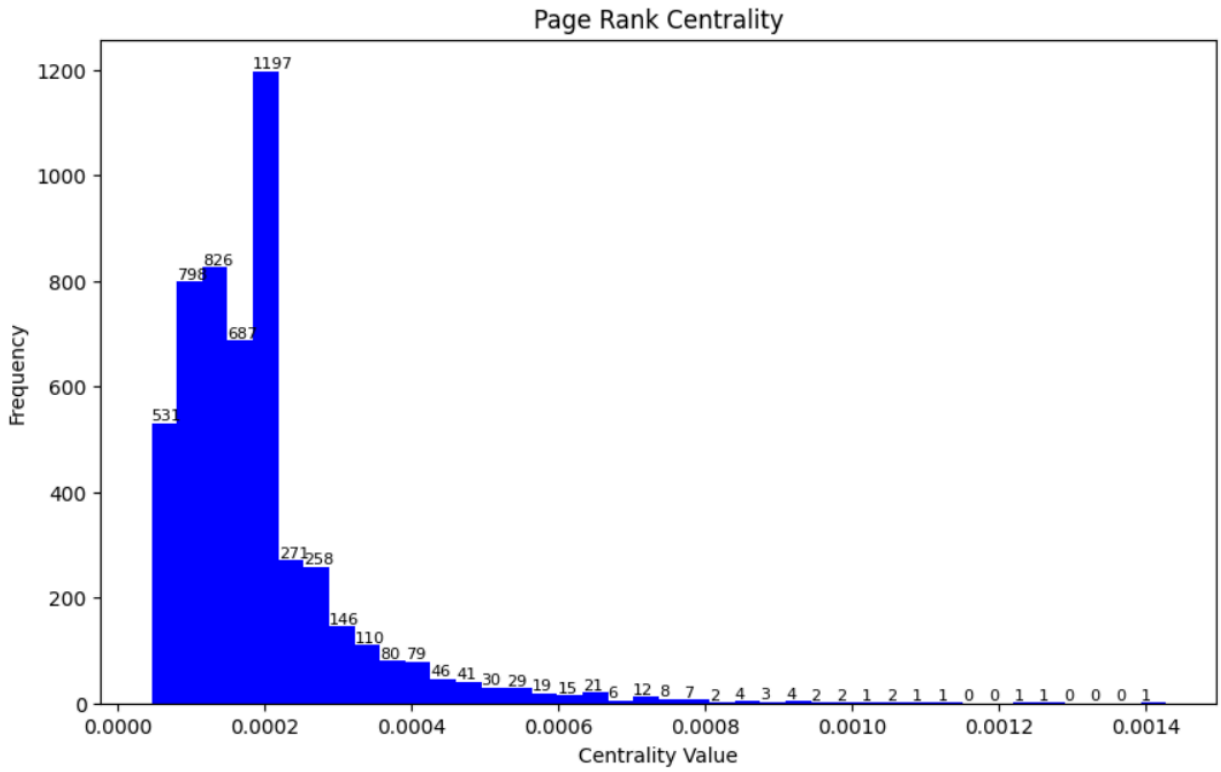


PageRank Centrality

Authors with a high PageRank Centrality value will likely get cited by other well-regarded authors. It reflects how much authors' work is referenced by others in the field, suggesting the reference and importance of their research. This helps identify prominent authors, can also be useful in reflecting how impactful the author is and finally helps identify emerging scholars.

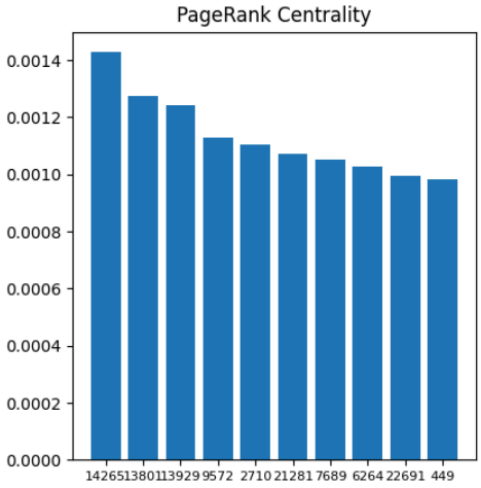
```
pr = nx.pagerank(G, alpha=0.8)

# Print each node and its Page Rank
for node, centrality in pr.items():
    print(f"Node {node}: Page Rank  = {centrality}")
```

```
# Plot distribution of centrality values for Page Rank Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Page Rank Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	PageRank Centrality
14265	0.001428
13801	0.001277
13929	0.001242
9572	0.001130
2710	0.001103
21281	0.001071
7689	0.001053
6264	0.001026
22691	0.000994
449	0.000982

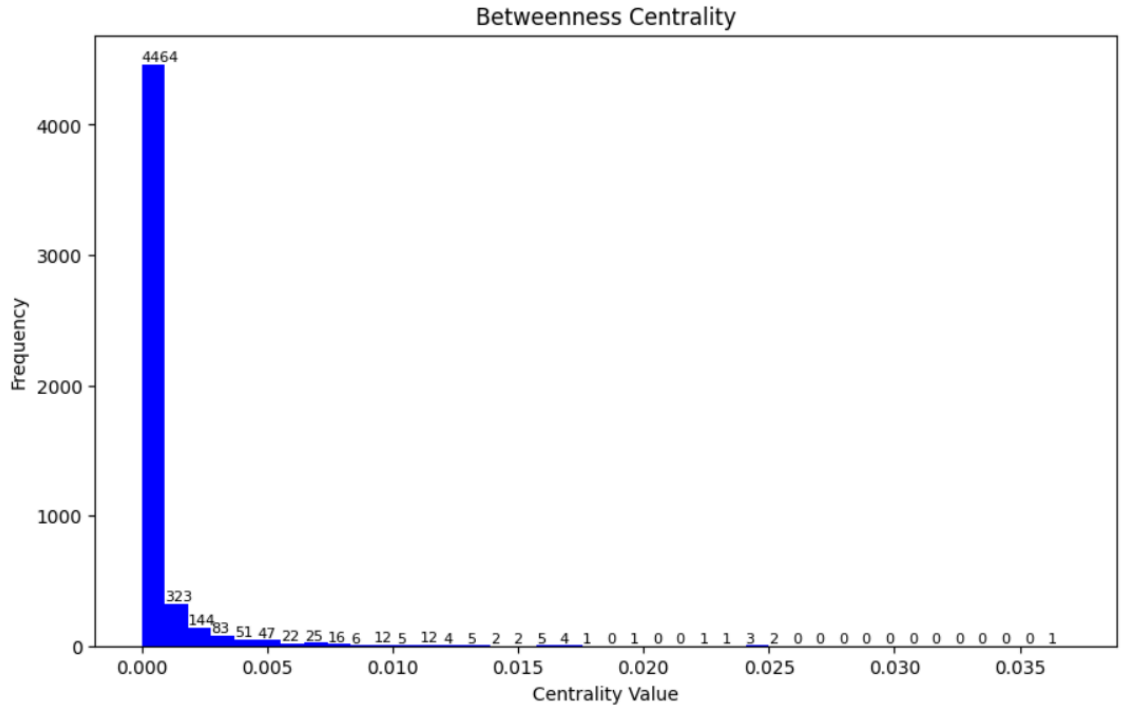


Betweenness Centrality

Betweenness here generally focuses on knowledge flow. Authors with high betweenness centrality might play a crucial role in facilitating the flow of knowledge and ideas between the authors. Authors with high betweenness centrality values will likely be more comfortable working across disciplinary boundaries.

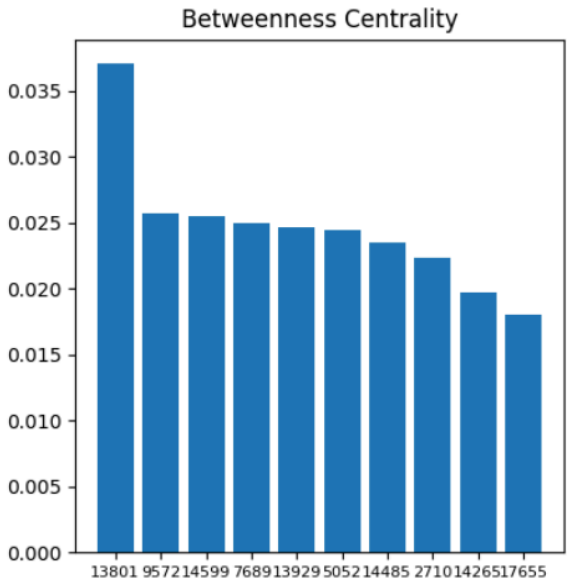
```python
bet_centrality = nx.betweenness_centrality(G, normalized = True,endpoints = False)

#Print each node and its Betweenness Centrality (excluding nodes with centrality = 0)
for node, centrality in bet_centrality.items():
    if centrality != 0:
        print(f"Node {node}: Betweenness Centrality = {centrality}")
```

```python
# Plot distribution of centrality values for Betweenness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Betweenness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	Betweenness Centrality
13801	0.037027
9572	0.025689
14599	0.025488
7689	0.024971
13929	0.024653
5052	0.024422
14485	0.023503
2710	0.022355
14265	0.019731
17655	0.017999

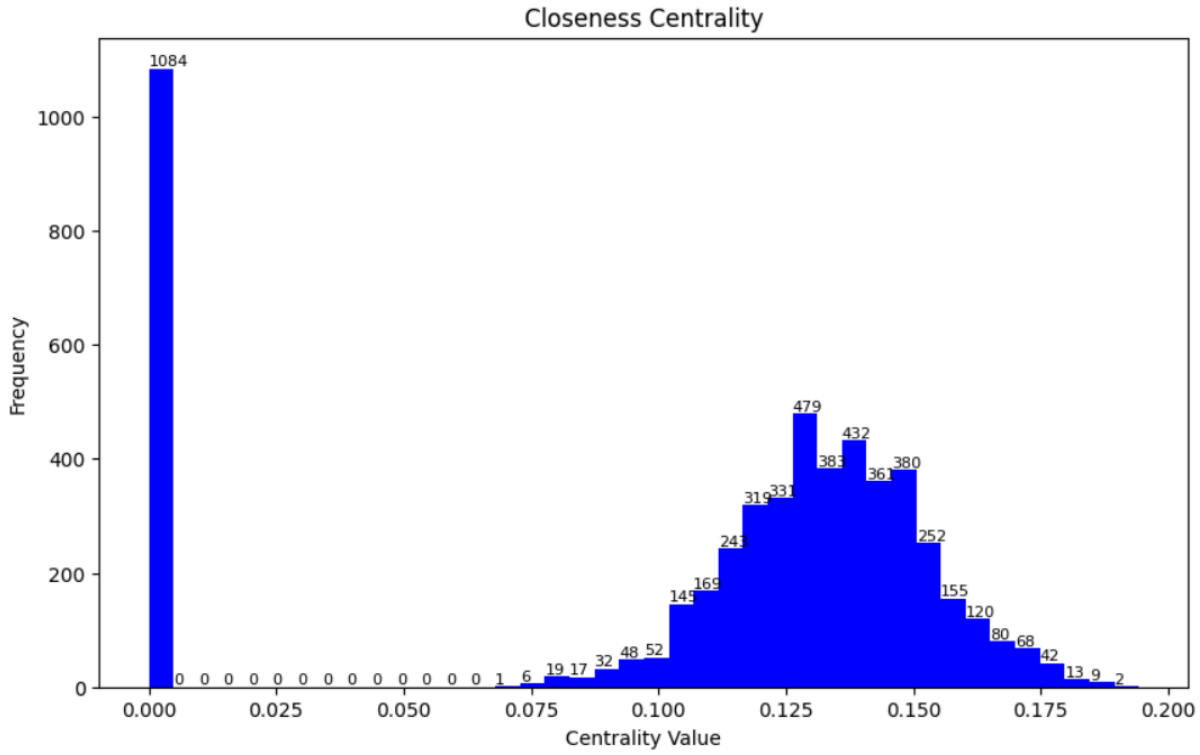


Closeness Centrality

The higher closeness centrality value shows that those authors are closer to all other authors in the network. Authors with high-value scores will likely be more central to the collaboration network. This particularly focuses on how easily reachable an author is from all other authors. A high closeness centrality indicates that, on average, an author is close to most other authors in the network

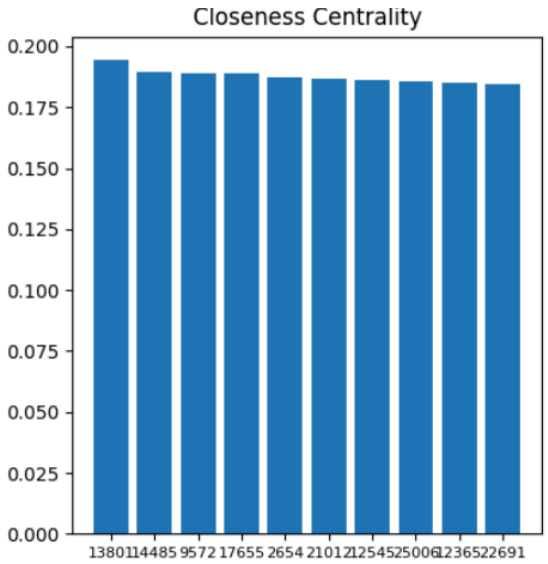
```
#Calculate Closeness centrality
close_centrality = nx.closeness_centrality(G)

#Print each node and its Closeness centrality
for node, centrality in close_centrality.items():
    print(f"Node {node}: Closeness centrality = {centrality}")
```

```
# Plot distribution of centrality values for Closeness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Closeness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



Top 10 Description

NODE	Closeness Centrality
13801	0.194285
14485	0.189538
9572	0.189038
17655	0.188962
2654	0.187129
21012	0.186546
12545	0.185989
25006	0.185601
12365	0.185309
22691	0.184759



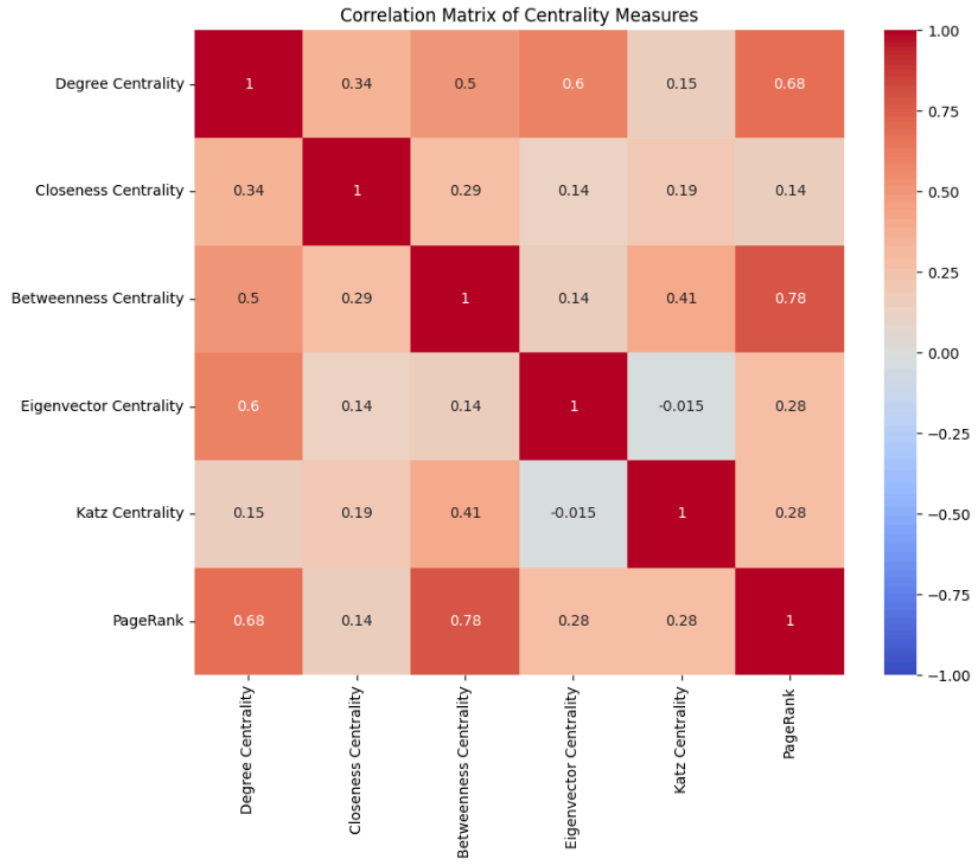
Correlation Matrix Between The Centrality Values

A high correlation between PageRank and Betweenness Centrality suggests that important nodes, according to PageRank, also tend to have high betweenness centrality, and vice versa. This could indicate that nodes that are important in their connections to other important nodes

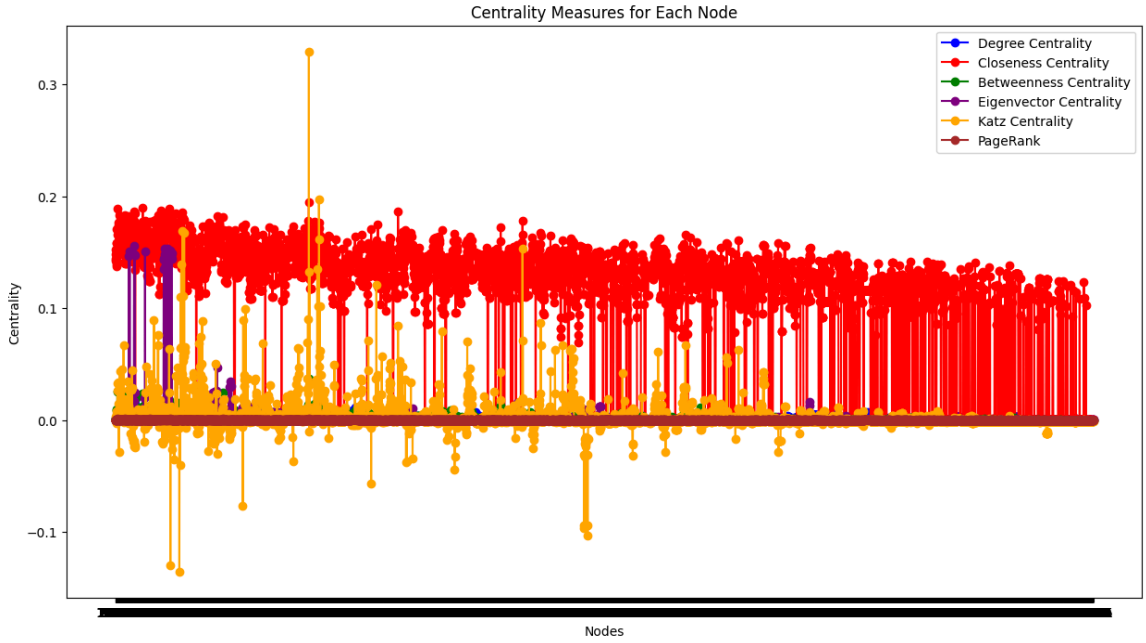
(PageRank) also tend to lie on many shortest paths between other nodes (Betweenness Centrality).

The negative value for eigenvector and Katz centrality suggests that these measures are insignificant in this network. Eigenvector centrality measures a node's influence based on its neighbours' influence. In contrast, Katz centrality measures a node's influence based on its neighbours' number and influence. The negative value indicates that these measures are not useful in identifying influential authors in this network.

On the other hand, the positive values for betweenness centrality and PageRank suggest that these measures are significant in this network. Betweenness centrality measures the extent to which a node lies on the shortest paths between other nodes, indicating its importance as a bridge or mediator in the network. PageRank is a measure of the importance of a node based on the number and quality of other nodes linking to it, indicating its overall importance in the network.
The degree centrality and PageRank scores of 0.68 suggest a moderate correlation between these two measures. Degree centrality measures the number of direct connections a node has, while PageRank measures a node's overall importance based on its neighbours' importance. A moderate correlation suggests that authors with many direct connections also have important neighbors, indicating their overall importance in the network.



Visualizing Centrality Measures



Clustering Coefficients

Global Clustering Coefficient

```
#Calculate Global Clustering Coefficient
global_clustering_coefficient = nx.transitivity(G)

#Print Global Clustering Coefficient
print("Global Clustering Coefficient:", global_clustering_coefficient)
```

```
Global Clustering Coefficient: 0.6298424741263426
```

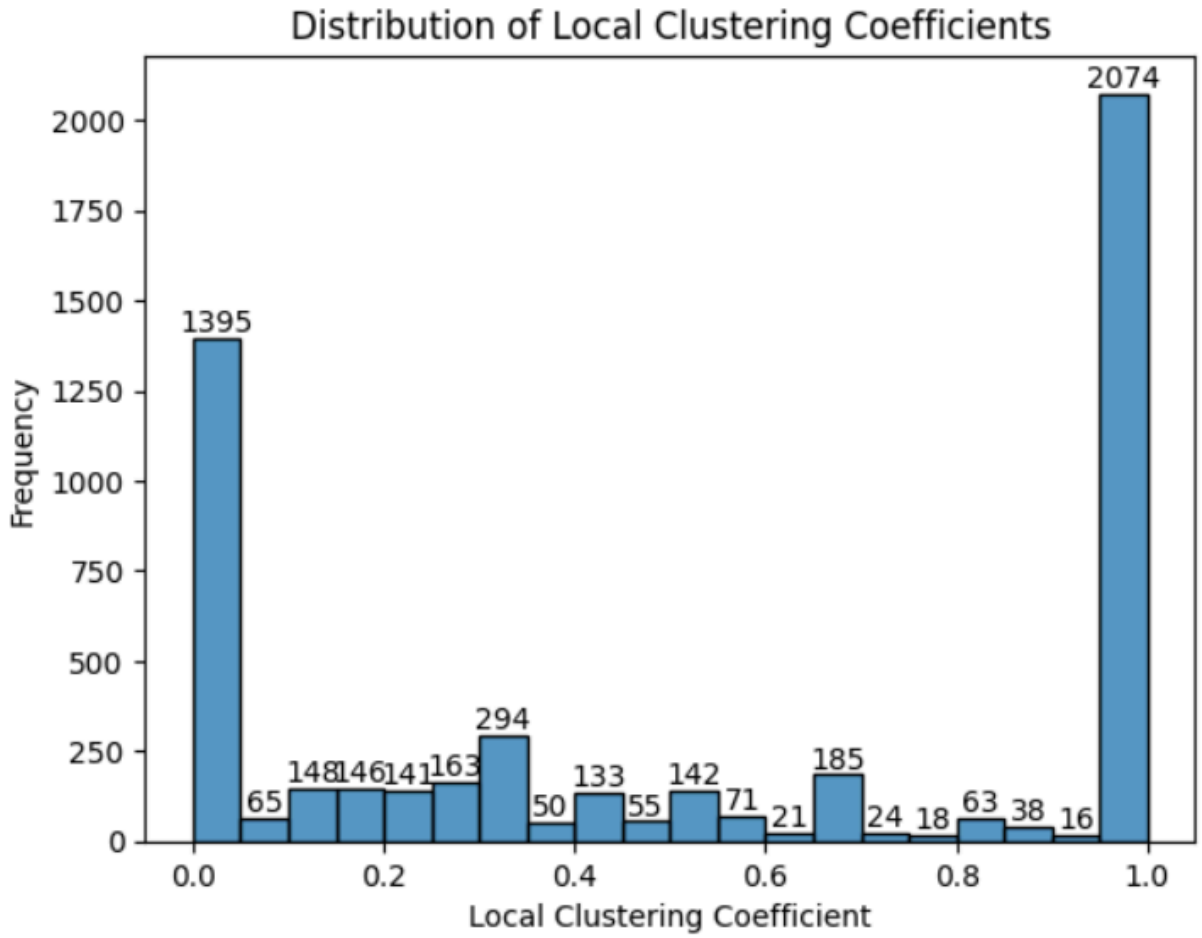
Local Clustering Coefficient

```
#Calculate Local Clustering Coefficient
local_clustering_coefficient = nx.clustering(G)

#Print each node and it's Clustering Coefficient
for node, local in local_clustering_coefficient.items():
    print(f"Node {node}: Local Clustering Coefficient = {local}")
```

```python
#Count the number of nodes with local clustering coefficient equal to 1
nodes_with_coefficient_1 = sum(1 for coeff in local_clustering_coefficient.values() if coeff == 1)
print(f"Number of Nodes with Local Clustering Coefficient 1: {nodes_with_coefficient_1}")
```

```
Number of Nodes with Local Clustering Coefficient 1: 2038
```

```python
#Create a line chart for the distribution of local clustering coefficients
coefficients = list(local_clustering_coefficient.values())
coefficients.sort()
unique_coefficients = list(set(coefficients))
counts = [coefficients.count(coeff) for coeff in unique_coefficients]

#Create a histogram for the distribution of local clustering coefficients
sns.histplot(list(local_clustering_coefficient.values()), bins=20, kde=False)
plt.xlabel('Local Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('Distribution of Local Clustering Coefficients')

#Display the frequency count on top of each bin
for p in plt.gca().patches:
    plt.gca().annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.get_height()),ha='center', va='bottom', color='black', fontsize=10)

plt.show()
```



Average Local Clustering Coefficient

```
#Calculate Average Local Clustering Coefficient
average_local_clustering_coefficient = nx.average_clustering(G)

print("Average Local Clustering Coefficient:", average_local_clustering_coefficient)
```

```
Average Local Clustering Coefficient: 0.529635811052136
```

Transitivity and Reciprocity

Transitivity

```
transitivity = nx.transitivity(G)
print("Transitivity:", transitivity)
```

```
Transitivity: 0.6298424741263426
```

Reciprocity

Reciprocity is a simplified version of transitivity because it considers closed loops of length 2, which can only happen in directed graphs. However, our chosen network, in this case, is undirected. Hence, we cannot find a value for the reciprocity of this network.

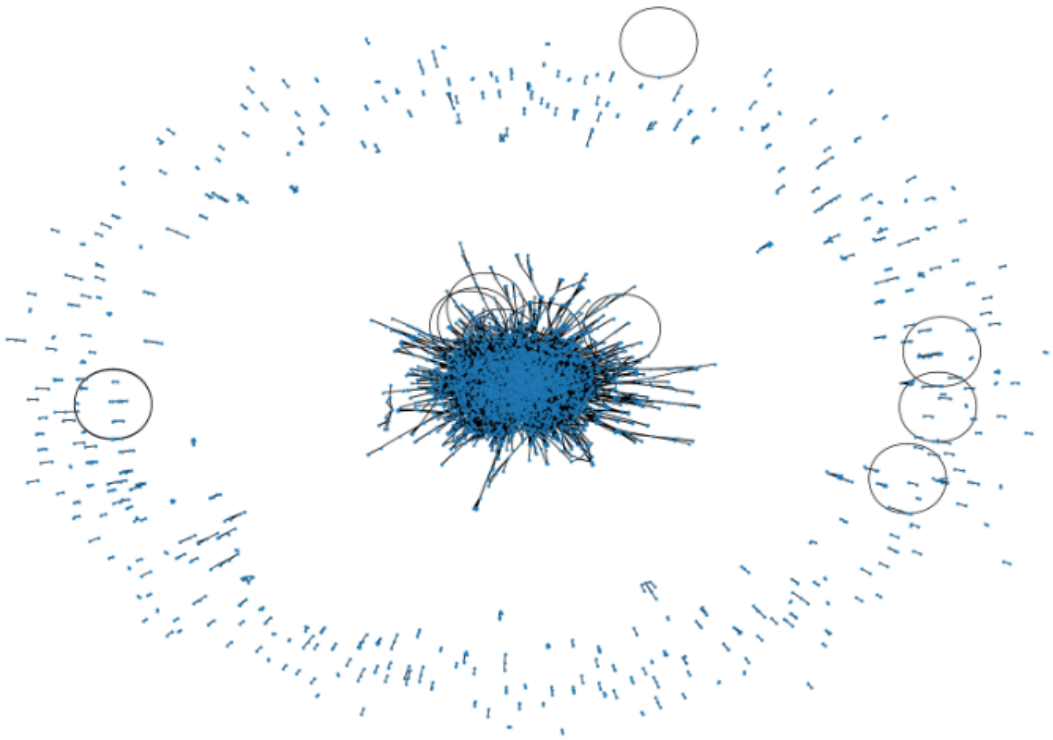
Largest Connected Component

```
#Find the largest connected component
largest_component = max(nx.connected_components(G), key=len)

#Create a subgraph of the largest connected component
largest_component_subgraph = G.subgraph(largest_component)

#Print the number of nodes and edges in the largest connected component
print("Number of nodes in the largest connected component:", largest_component_subgraph.number_of_nodes())
print("Number of edges in the largest connected component:", largest_component_subgraph.number_of_edges())

#Visualize the largest connected component
plt.figure(figsize=(18, 12))
nx.draw(largest_component_subgraph, with_labels=False, node_size = 10)
plt.title('Largest Connected Component')
plt.show()
```

```
Number of nodes in the largest connected component: 4158
Number of edges in the largest connected component: 13428
```

Complete Network Visualization

```
plt.figure(figsize=(18, 12))
nx.draw(G, with_labels=False, node_size = 5, width = 1)
plt.show()
```



Comparative Analysis Between The Two Networks

Centrality Measures From Network - 1

```
#Compare centrality measures based on criteria

mean_degree = sum(deg_centrality.values()) / len(deg_centrality)
mean_closeness = sum(close_centrality.values()) / len(close_centrality)
mean_betweenness = sum(bet_centrality.values()) / len(bet_centrality)
mean_eigenvector = sum(eigenvector_centrality.values()) / len(eigenvector_centrality)
mean_katz = sum(katz_centrality.values()) / len(katz_centrality)
mean_pagerank = sum(pr.values()) / len(pr)

print("Mean Degree Centrality:", mean_degree)
print("Mean Closeness Centrality:", mean_closeness)
print("Mean Betweenness Centrality:", mean_betweenness)
print("Mean Eigenvector Centrality:", mean_eigenvector)
print("Mean Katz Centrality:", mean_katz)
print("Mean PageRank:", mean_pagerank)
```

```
Mean Degree Centrality: 0.002057944872638308
Mean Closeness Centrality: 0.04566886623059432
Mean Betweenness Centrality: 0.011629407659036898
Mean Eigenvector Centrality: 0.007259108836993484
Mean Katz Centrality: 0.028890194343833876
Mean PageRank: 0.0008517887563884146
```

Centrality Measures From Network - 2

```
#Compare centrality measures based on criteria

mean_degree = sum(deg_centrality.values()) / len(deg_centrality)
mean_closeness = sum(close_centrality.values()) / len(close_centrality)
mean_betweenness = sum(bet_centrality.values()) / len(bet_centrality)
mean_eigenvector = sum(eigenvector_centrality.values()) / len(eigenvector_centrality)
mean_katz = sum(katz_centrality.values()) / len(katz_centrality)
mean_pagerank = sum(pr.values()) / len(pr)

print("Mean Degree Centrality:", mean_degree)
print("Mean Closeness Centrality:", mean_closeness)
print("Mean Betweenness Centrality:", mean_betweenness)
print("Mean Eigenvector Centrality:", mean_eigenvector)
print("Mean Katz Centrality:", mean_katz)
print("Mean PageRank:", mean_pagerank)
```

```
Mean Degree Centrality: 0.001055278280507871
Mean Closeness Centrality: 0.10630130466095028
Mean Betweenness Centrality: 0.0006062707858585091
Mean Eigenvector Centrality: 0.0016145570004595063
Mean Katz Centrality: 0.001991474673439358
Mean PageRank: 0.0001907668828691324
```

Inferences from the various centrality measures of both networks

Degree Centrality
The mean degree centrality of network 1 is 0.0021, while that of network 2 is 0.0011. This suggests that nodes are, on average, more connected in network 1.

Closeness Centrality
The mean closeness centrality of network 1 is 0.0457, while that of network 2 is 0.1063. This suggests that nodes are much closer to each other on average in network 1, indicating a more tightly connected network.

Betweenness Centrality

The mean betweenness centrality of network 1 is 0.0116, while that of network 2 is 0.0006. This suggests that nodes in network 1 play a more critical role in connecting other nodes.

Eigenvector Centrality

The mean eigenvector centrality of 0.0073 suggests that nodes in network 1 have, on average, moderate influence or importance within the network. The mean eigenvector centrality of 0.0016 in network 2 indicates that nodes have a lower influence or importance than network 1.

Katz Centrality

The mean Katz centrality of 0.0289 suggests that nodes in network 1 have, on average, moderate influence or importance considering both direct and indirect connections.
network 2: The mean Katz centrality of 0.0020 in network 2 indicates that nodes have a lower influence or importance than network 1.

PageRank
The higher mean PageRank of 0.0009 suggests that nodes in network 1 are, on average, more important or influential according to the PageRank algorithm. The lower mean PageRank of 0.0002 in network 2 indicates that nodes are less essential or influential according to the PageRank algorithm compared to network 1.

Conclusion

In conclusion, network 1 (Euroroads) (Euroroads) has a more tightly connected network with nodes with more connections and closer proximity, playing a more critical role in connecting other nodes. Nodes in network 1 also have more influence or importance within the network, considering both direct and indirect connections and the PageRank algorithm. Therefore, network 1 has a more complex and interconnected structure than network 2 (General Relativity and Quantum Cosmology Collaboration Network).

If you wish to check the code for the project, refer to the following GitHub link:
https://github.com/RonitGupta2002/SNA-Project-Network-Analysis

SOCIAL NETWORK ANALYSIS

Network Analysis (Round 2)

Tasks Done:

Generating And Analyzing A Barabasi-Albert Network
Comparative Analysis Between The Networks Used (Both Round 1 and 2)
Implementing The Independent Cascade Model(ICM)

Project Report By Team Alpha

Ronit Gupta(21UCS173)

Ruthvik Devavarapu (21UCS174)

Saumya Saraswat(21UCS184)

Saunak Kushwaha(21UCS185)

Course Instructor

Dr. Sakthi Balan

Table Of Contents

Problem Statement

The second round has two problems:

A. Generate BA Network:
   a. Implement the BA Algorithm to generate the scale-free network S over 10000 nodes. You may assume the initialization (for BA) as per your wish but adhering to the characteristics laid about by the BA Model. State your initialization clearly.
   b. Do all the studies conducted for Project Round 1 for the network S and compare and contrast the values between the previous networks and S.
   c. Find the Giant component for all three networks and compare and contrast the basic statistics of the networks.

B. Information Diffusion in the Networks:
   a. Apply ICM (Independent Cascade Model) to find the number of steps required to get to the maximum number of nodes on all the three networks.
   b. This you may repeat 10 times by starting from different nodes in each networks and see what is the average number of steps required.
   c. Activation probabilities for the pair of nodes which is needed for ICM can be assigned randomly. When you are assigning it randomly note this point: from a node say v if there are three edges to different vertices w,x, and y. Then it should be $p(v,w)+p(v,x)+p(v,y)=1$.

Once you do all of the above, report your findings in a document.

Tools Used

```
#Libraries for Data Manipulation
import numpy as np
import pandas as pd

#Libraries for Graph Plotting
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

#Libraries for Network Analysis
import networkx as nx
from networkx import empty_graph, star_graph

import random
```

Besides using some standard Data Manipulation and Visualization packages, we have used **NetworkX,** a Python package, to create, manipulate, and study complex networks' structure, dynamics, and functions to perform the required network analysis in our project.

About NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It provides:

- Tools for the study of the structure and dynamics of social, biological, and infrastructure networks;
- a standard programming interface and graph implementation that is suitable for many applications;
- a rapid development environment for collaborative, multidisciplinary projects;
- an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and
- The ability to work painlessly with large nonstandard data sets.

With NetworkX, you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyse network structure, build network models, design new network algorithms, draw networks, and more.

Generating a Barabasi Albert Network

Algorithm For Generating The Network

The function below is a utility function that returns *"m"* unique elements from a given sequence (Our network of pre-existing nodes in this case). This function differs from the *random.sample()* because the selected subsequence of *"m"* nodes will not have any repeated elements. The nodes from this subsequence are considered our target nodes for implementing preferential attachment, a vital feature of the Barabasi-Albert Model.

Note: *Here: 'rng' is a random.Random or numpy.random.RandomState instance.*

```
def random_subset(seq, m, rng):
    # Create an empty set to store the unique elements
    targets = set()

    # Loop until we have m unique elements
    while len(targets) < m:
        # Choose a random element from the sequence
        x = rng.choice(seq)
        # Add the chosen element to the set
        targets.add(x)

    # Return the set of m unique elements
    return targets
```

After this, we define a function to generate the needed Barabasi-Albert network. The parameters of this function are as follows-

- **n:** The number of nodes we need in our final generated graph is **10,000** in our network.
- **m:** The number of initial nodes in the network, we have decided this to be **2** for our network.
- **seed:** By using a seed, we can generate reproducible Barabasi-Albert networks. Setting a seed value allows peers to assess and analyse the network efficiently. The seed value for our network is **1**.

The function generates our required Barabasi-Albert network, starting with a star graph of *"m"* nodes and returning the generated network as the function output.

```python
def barabasi_albert_graph(n, m, seed=None, initial_graph=None):
    # Validate the input parameters
    if m < 1 or m >= n:
        print(f"Barabási-Albert network must have m >= 1 and m < n, m = {m}, n = {n}")

    # Create the initial graph
    G = star_graph(m)

    # Set up the random number generator
    rng = random.Random(seed) if seed is not None else random.Random()

    # Create a list of repeated nodes based on their degree
    repeated_nodes = [n for n, d in G.degree() for _ in range(d)]
    source = len(G)

    # Add new nodes and edges to the graph
    while source < n:
        # Select m unique target nodes using the random_subset function
        targets = random_subset(repeated_nodes, m, rng)
        # Add the new edges to the graph
        G.add_edges_from(zip([source] * m, targets))
        # Update the list of repeated nodes
        repeated_nodes.extend(targets)
        repeated_nodes.extend([source] * m)
        source += 1

    return G
```

In the Barabási–Albert model, when a new node is added to the graph, it connects to m existing nodes. The probability of connecting to an existing node is proportional to its degree. Nodes with higher degrees are likelier to be selected as connection targets, mimicking the "rich get richer" principle observed in many real-world networks.

By creating a list of repeated nodes based on their degree, the code effectively creates a pool of potential target nodes for new edges. Nodes with higher degrees will appear more times in the repeated_nodes list, increasing their chances of being selected as connection targets for new nodes. This process helps generate a scale-free network topology characteristic of many real-world networks. We then simply call the function as shown below to generate our network.

```python
S = barabasi_albert_graph(10000, 2, 1)
for edge in S.edges():
    print(edge)
```

Network Statistics

```
print('Nodes:', len(S.nodes))
print('Edges:', len(S.edges))
max_degree = max(degrees)
min_degree = min(degrees)
avg_degree = np.mean(degrees)
std_degree = np.std(degrees)

print("Max Degree of a Node:", max_degree)
print("Min Degree of a Node:", min_degree)
print("Average Degree of Nodes:", avg_degree)
print("Standard Deviation of Degree Distribution:", std_degree)
```

```
Nodes: 10000
Edges: 19996
Max Degree of a Node: 264
Min Degree of a Node: 2
Average Degree of Nodes: 3.9992
Standard Deviation of Degree Distribution: 6.53397270885026
```

The above output displays some standard network statistics for our generated Barabasi-Albert network

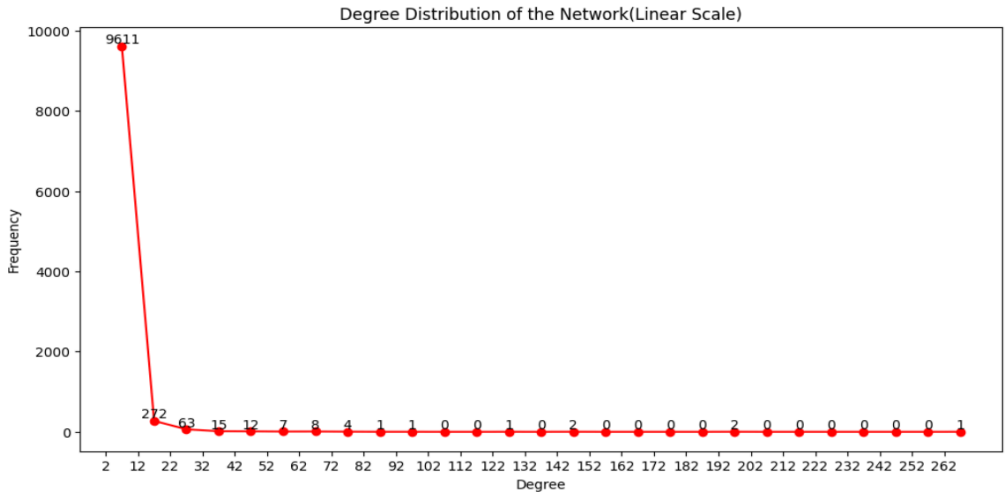
---

.

Degree Distribution

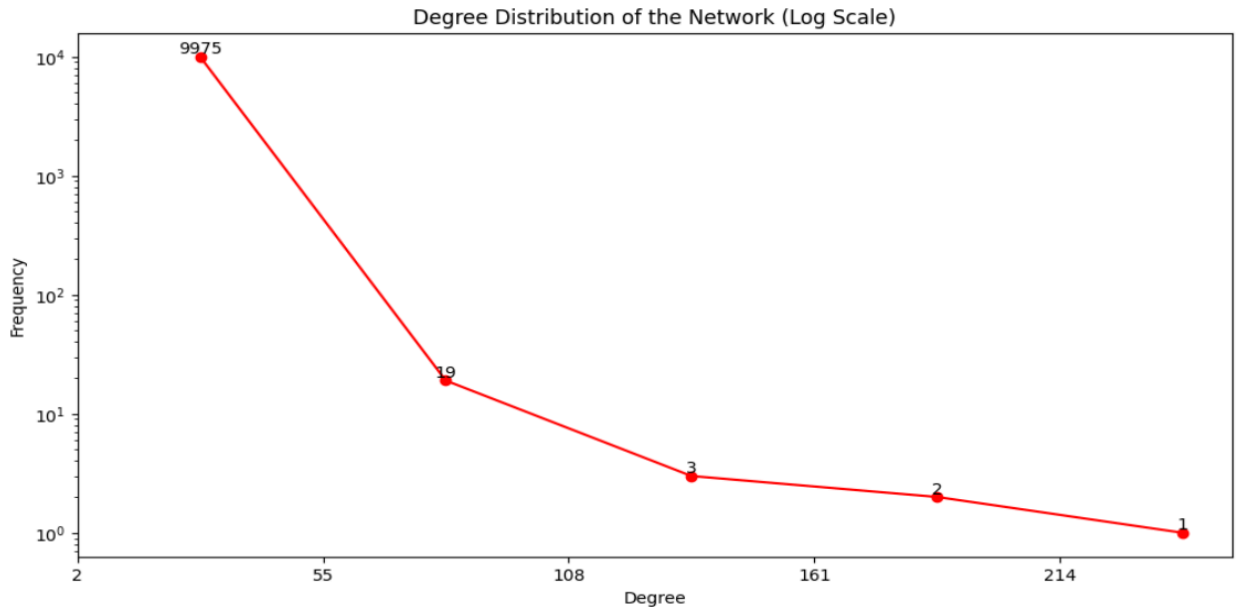
```python
# Define the desired bin size
bin_size = 10

# Determine the range of degrees and calculate the number of bins
degree_range = max(degrees) - min(degrees)
num_bins = int(degree_range / bin_size) + 1

# Define the bins based on the calculated number of bins
bins = range(min(degrees), max(degrees) + bin_size, bin_size)

# Create Histogram
hist, bins = np.histogram(degrees, bins=bins)

# Calculate the centers of the bins for the line plot
bin_centers = 0.5 * (bins[:-1] + bins[1:])

# Create line plot
plt.figure(figsize=(12, 6))
plt.plot(bin_centers, hist, linestyle='-', marker='o', color='red')

# Add Text Labels for Bin
for i, bin_value in enumerate(bins[:-1]):
    plt.text(bin_value + bin_size / 2, hist[i], str(hist[i]), ha='center', va='bottom')

# Set x-axis tick locations and labels
plt.xticks(range(min(degrees), max(degrees) + 1, bin_size))

# Set labels and title
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the Network(Linear Scale)')

# Show plot
plt.show()
```

We use the above code to plot a degree distribution for our generated network with a *bin_size* of **10** for convenience. Doing this makes the plot more legible and easier to understand.



We also generated this distribution on a logarithmic scale with the code below, setting our *bin_size* to **53** for the mentioned reasons.

```
# Define the desired bin size
bin_size = 53

# Determine the range of degrees and calculate the number of bins
degree_range = max(degrees) - min(degrees)
num_bins = int(degree_range / bin_size) + 1

# Define the bins based on the calculated number of bins
bins = range(min(degrees), max(degrees) + bin_size, bin_size)

# Create Histogram
hist, bins = np.histogram(degrees, bins=bins)

# Calculate the centers of the bins for the line plot
bin_centers = 0.5 * (bins[:-1] + bins[1:])

# Create line plot
plt.figure(figsize=(12, 6))
plt.plot(bin_centers, hist, linestyle='-', marker='o', color='red')

# Add Text Labels for Bin
for i, bin_value in enumerate(bins[:-1]):
    plt.text(bin_value + bin_size / 2, hist[i], str(hist[i]), ha='center', va='bottom'

# Set x-axis tick locations and labels
plt.xticks(range(min(degrees), max(degrees) + 1, bin_size))

# Set y-axis scale to logarithmic
plt.yscale('log')

# Set labels and title
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the Network (Log Scale)')

# Show plot
plt.show()
```



The Barabasi-Albert network showcases a preferential attachment mechanism, where the newly introduced nodes in a scale-free network prefer to attach themselves with nodes of higher degrees. This results in the rise of a few nodes with excessively high degrees also referred to as *"hubs"*. This is also reflected in the above degree distributions, which show a small number of nodes with an extremely high degree.

However, there are also many nodes with an extremely low degree of degree due to this mechanism; a newer node with fewer connections is less likely to be selected by the preferential attachment mechanism due to its low degree. This phenomenon is also very evident in the degree distribution shown above.

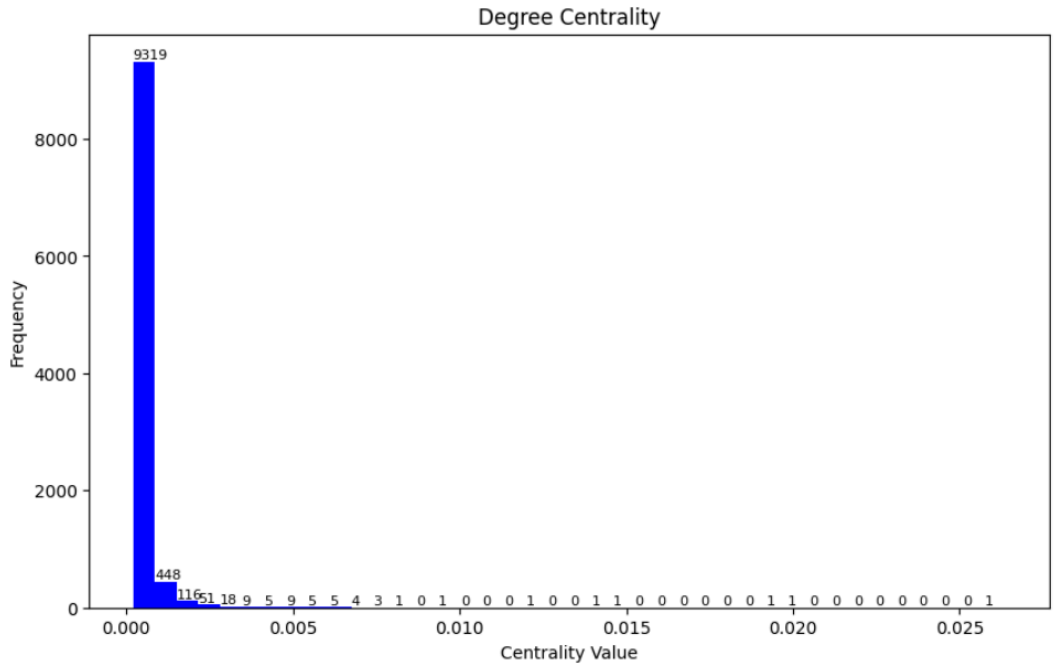
Centrality Measures

Degree Centrality

The code below prints a plot of the degree centrality of the various nodes in the graph, with a **bin size of 40**, to make the graph easier to read and understand.

```
#Calculate Degree Centrality
deg_centrality = nx.degree_centrality(G)

#Print each node and its Degree Centrality
for node, centrality in deg_centrality.items():
    print(f"Node {node}: Degree Centrality = {centrality}")
```

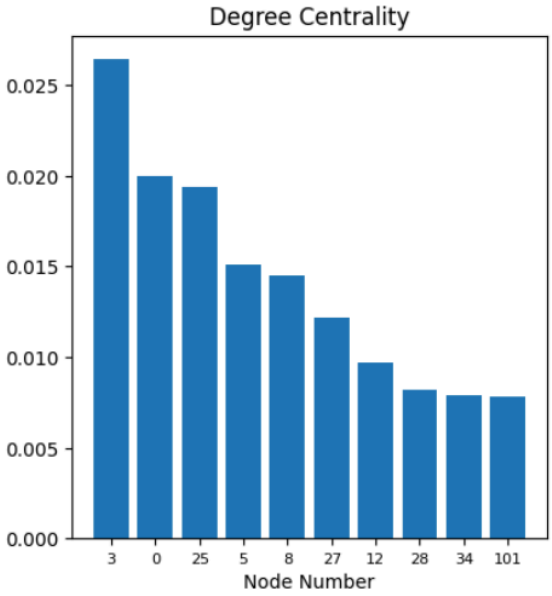
```
# Plot distribution of centrality values for Degree Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Degree Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(deg_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



As a direct consequence of the preferential attachment mechanism, we also see many nodes with a low degree centrality, entirely in line with our understanding of the Barabasi-Albert Model. At the same time, the number of nodes with an extremely high value of degree centrality (Compared to other nodes) is very low and is seen only when a node is a hub in the network.

Top 10 Description

NODES	DEGREE CENTRALITY
3	0.026403
0	0.020002
25	0.019402
5	0.015102
8	0.014501
27	0.012201
12	0.009701
28	0.008201
34	0.007901
101	0.007801

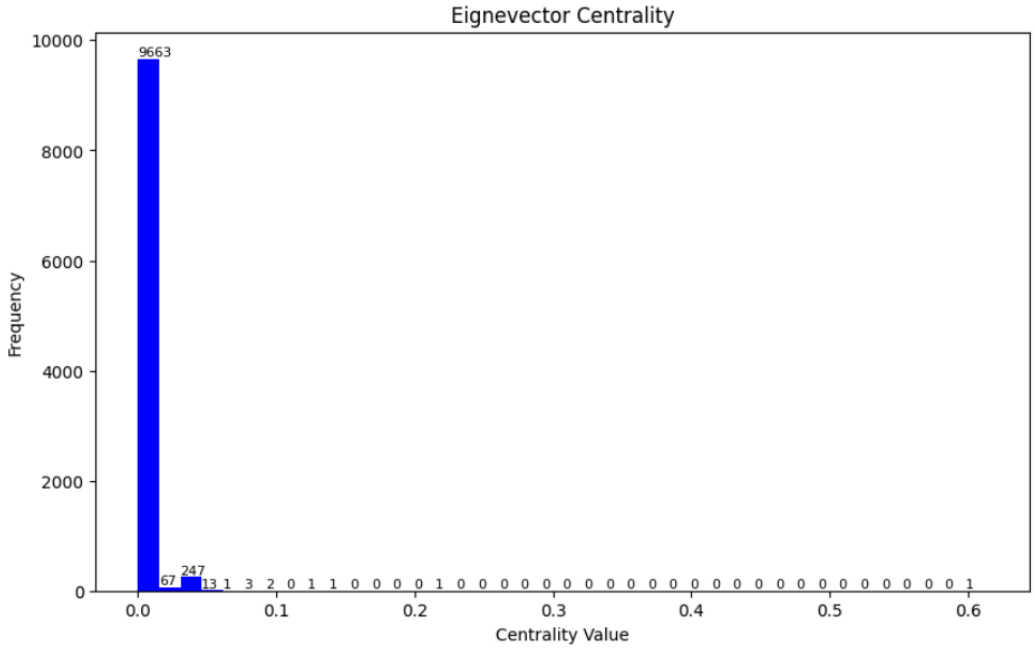


Eigenvector Centrality

The code below prints a plot of the eigenvector centrality of the various nodes in the graph, with a **bin size of 40**, to make the graph easier to read and understand.

```
#Calculate Eignevector Centrality
eigenvector_centrality = nx.eigenvector_centrality(G)

#Print each node and its Eignevector Centrality
for node, centrality in eigenvector_centrality.items():
    print(f"Node {node}: Eignevector Centrality = {centrality}")
```

```
# Plot distribution of centrality values for Eignevector Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Eignevector Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(eigenvector_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



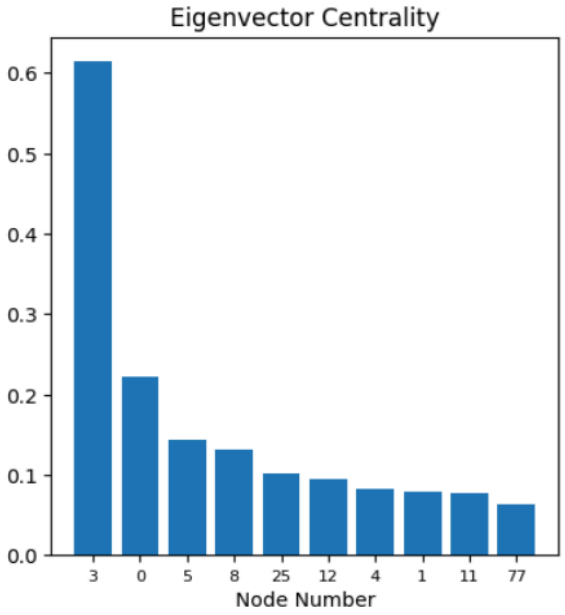
As a direct consequence of the preferential attachment mechanism, we also see that most nodes have a very low eigenvector centrality value, indicating that most links between a node connect it to a node that isn't very well connected. There are a tiny number of hubs compared to the rest of the graph, so most connections are made to non-hub nodes, lowering the eigenvector centrality for most nodes.

However, a smaller group of nodes appears to have a relatively high eigenvector centrality. These are the nodes whose primary connections are to the hubs only. When a node is primarily

connected to hubs only, the highly connected nature of these hubs lends to a higher eigenvector centrality measure for these nodes.

Top 10 Description

NODES	EIGENVECTOR CENTRALITY
3	0.613823
0	0.22667
5	0.144317
8	0.131763
25	0.101004
12	0.094033
4	0.082203
1	0.079836
11	0.077949
77	0.063652

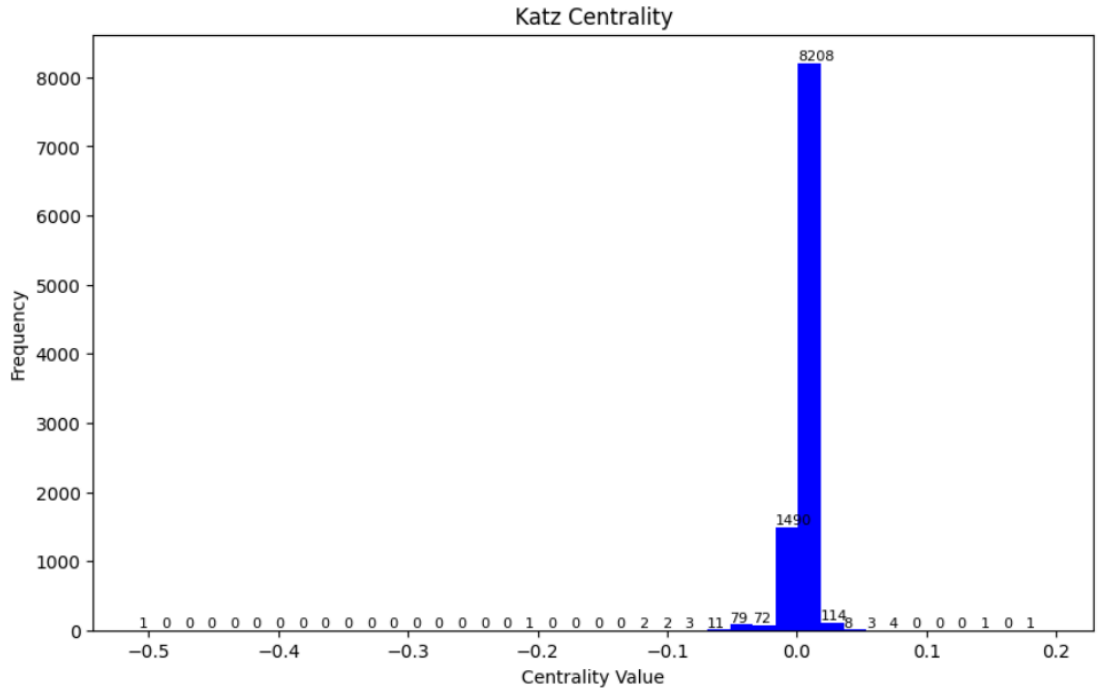


Katz Centrality

Katz centrality generally infers the potential influence of a node in the network based on the length of the paths connecting to other nodes. A node with many short paths leading to it is likely to have a greater influence when compared to one with fewer or longer paths. Katz Centrality can also be used to identify the key players in spreading information or ideas through the collaboration network.

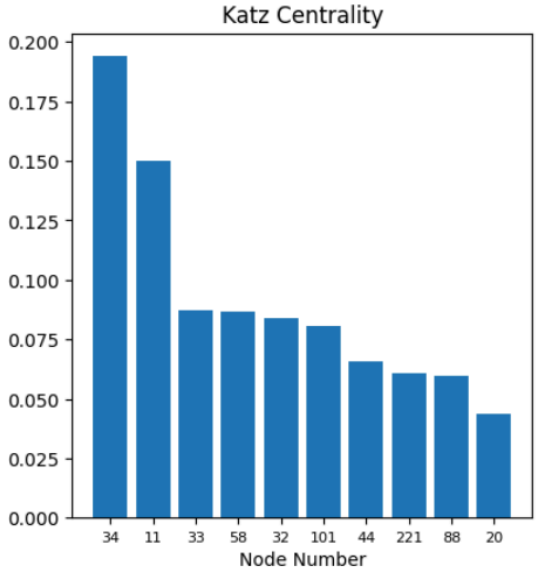
```
katz_centrality = nx.katz_centrality_numpy(G, alpha=0.1, beta=1.0, normalized=True, weight=None)

# Print each node and its Katz Centrality
for node, centrality in katz_centrality.items():
    print(f"Node {node}: Katz Centrality = {centrality}")
```

```
# Plot distribution of centrality values for Katz Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Katz Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(katz_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



As seen in the graph above, there are only a few nodes with a high measure for Katz Centrality; meanwhile, most nodes seem to have a low value for this measure. It indicates that there are only a few key players in this network for controlling the spread of information, which would be the hub nodes, in this case, who have a higher value for Katz Centrality. Top 10 Description

NODES	KATZ CENTRALITY
34	0.193930
11	0.149827
33	0.087218
58	0.086793
32	0.084070
101	0.080841
44	0.065915
221	0.060786
88	0.059829
20	0.043569

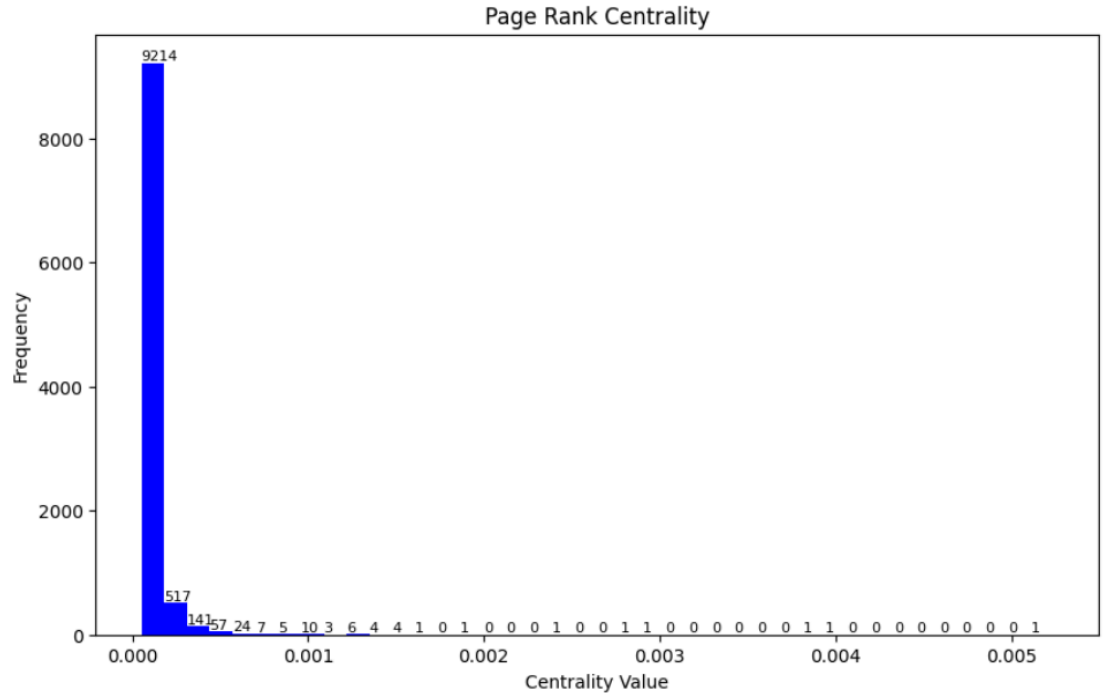


PageRank Centrality

Nodes with a high PageRank Centrality value will likely be connected to other nodes with a high PageRank centrality value. It reflects how important a node is to the network. This helps identify prominent nodes and other emerging nodes. This is because prominent nodes are more likely to be interconnected.

```python
pr = nx.pagerank(G, alpha=0.8)

# Print each node and its Page Rank
for node, centrality in pr.items():
    print(f"Node {node}: Page Rank  = {centrality}")
```

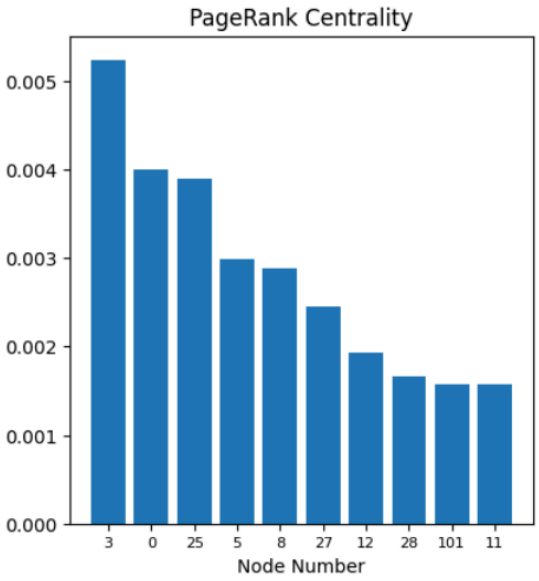
```python
# Plot distribution of centrality values for Page Rank Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Page Rank Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(pr.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



The graph above again aligns with our understanding of the Barabasi-Albert model. It shows how very few nodes have a high value for PageRank centrality. This means that very few nodes are prominent and connected to other prominent nodes. This is also why a significant fraction of the nodes have a low value for this measure. However, some nodes with slightly higher PageRank centrality values may be considered potential hubs as the network grows.

Top 10 Description

NODES	PAGERANK CENTRALITY
3	0.005238
0	0.003995
25	0.003887
5	0.002992
8	0.002880
27	0.002448
12	0.001926
28	0.001663
101	0.001576
11	0.001571

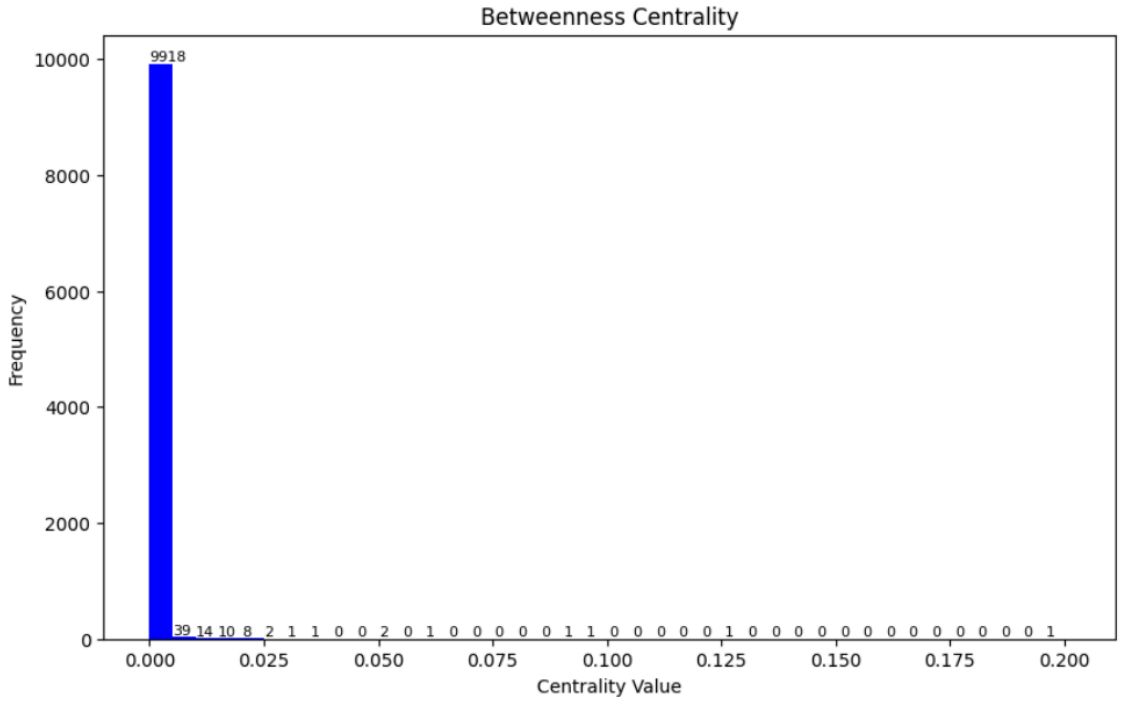


Betweenness Centrality

Betweenness here generally focuses on information flow. Nodes with high betweenness centrality might play a crucial role in facilitating the flow of information within the network. Nodes with high betweenness centrality values are likely to be associated with multiple sparsely connected groups of nodes and play a vital role in spreading information within a network.

```
bet_centrality = nx.betweenness_centrality(G, normalized = True,endpoints = False)

#Print each node and its Betweenness Centrality (excluding nodes with centrality = 0)
for node, centrality in bet_centrality.items():
    if centrality != 0:
        print(f"Node {node}: Betweenness Centrality = {centrality}")
```

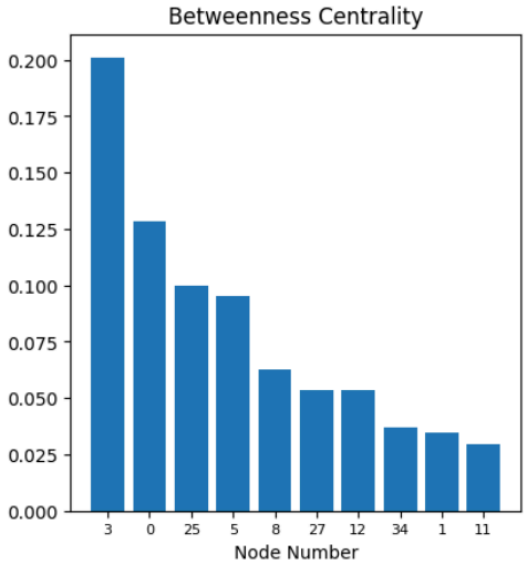
```
# Plot distribution of centrality values for Betweenness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Betweenness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(bet_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



The observations obtained above are again in line with our understanding of the Barabasi-Albert model. The presence of hubs in the network also makes them the de-facto nodes with a high betweenness similarity due to the nature of the networks generated using this network. The hubs connect otherwise disjoint and sparsely connected groups of nodes within the network effectively. Hence, the network's hubs have a high value for betweenness similarity, while most other nodes have a far lower value for this measure, as seen above.

Top 10 Description

NODES	BETWEENNESS CENTRALITY
3	0.201154
0	0.128382
25	0.100050
5	0.095491
8	0.062595
27	0.053470
12	0.053428
34	0.036765
1	0.034816
11	0.029575

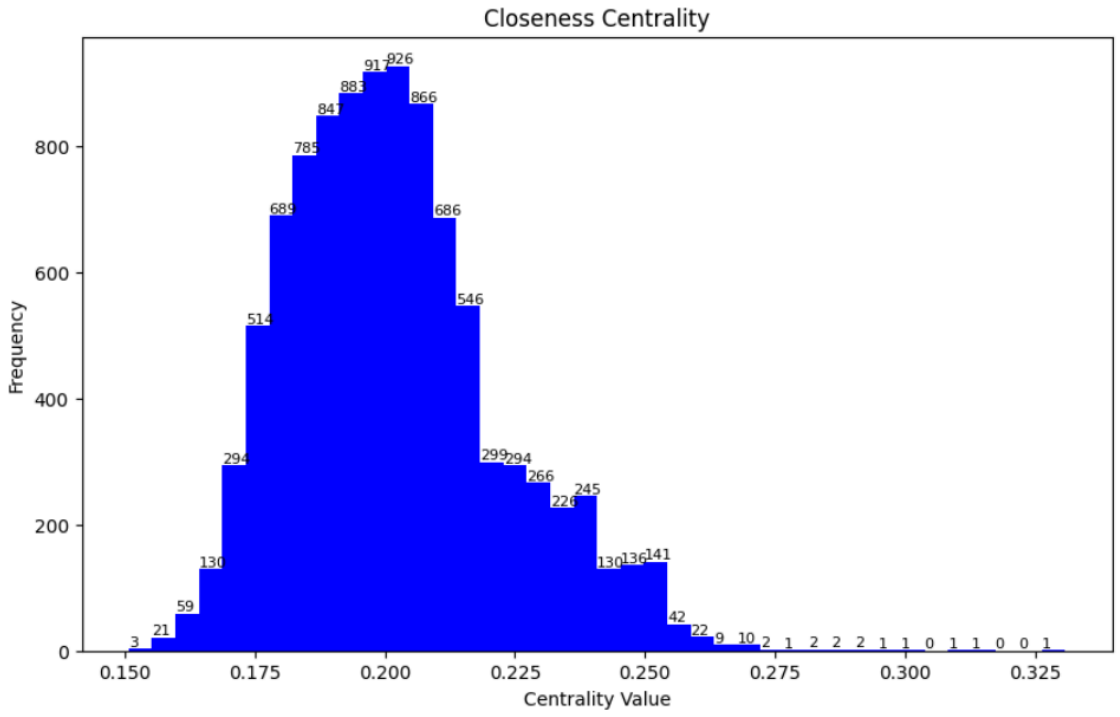


Closeness Centrality

The higher closeness centrality value shows that those nodes are closely linked to the other nodes in the network. Nodes with high-value scores will likely be more central to this network, making hubs the de facto nodes with a high value for this measure. This mainly focuses on how easily reachable a node is from all other nodes. A high closeness centrality indicates that, on average, a node is close to most other nodes in the network.

```
#Calculate Closeness centrality
close_centrality = nx.closeness_centrality(G)

#Print each node and its Closeness centrality
for node, centrality in close_centrality.items():
    print(f"Node {node}: Closeness centrality = {centrality}")
```

```
# Plot distribution of centrality values for Closeness Centrality
plt.figure(figsize=(10, 6))
plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)  # Increase bin size to 40
plt.title("Closeness Centrality")
plt.xlabel('Centrality Value')
plt.ylabel('Frequency')

# Write frequency on top of each bin
for i in range(40):
    plt.text(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[1][i],
             plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i],
             str(int(plt.hist(list(close_centrality.values()), bins=40, color='b', alpha=0.7)[0][i])),
             fontsize=8, ha='left', va='bottom')

plt.show()
```



In a Barabási–Albert (BA) network, nodes are added sequentially, and each new node is connected to m existing nodes based on preferential attachment, where nodes with higher

degrees are more likely to receive new connections. This preferential attachment mechanism tends to create a network with a scale-free degree distribution, where a few nodes have a very high degree while most nodes have a relatively low degree.

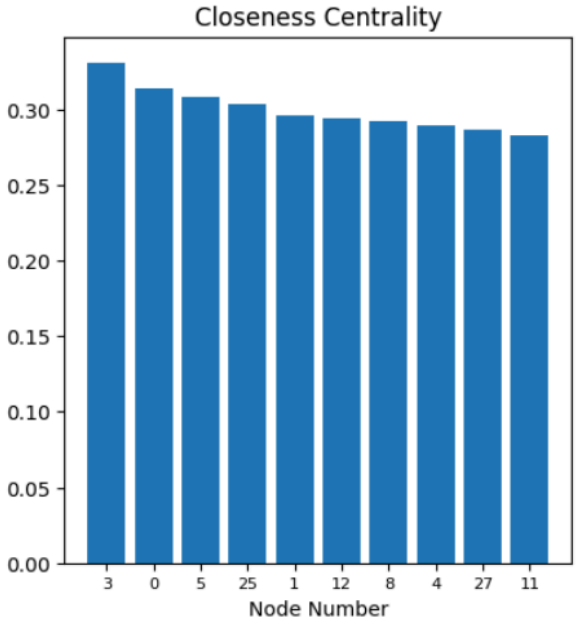
The closeness centrality of a node in a network measures how close it is to all other nodes in the network. It is calculated as the reciprocal of the average shortest path length from the node to all other nodes in the network.

In a BA network, because of the scale-free nature of the degree distribution, there tend to be a few highly connected "hub" nodes that are very close to most other nodes in the network. As a result, the closeness centrality of these hub nodes is high, and the distribution of closeness centrality values across nodes is likely to be uniform or nicely distributed. This is because most nodes can reach one of these hub nodes relatively quickly along short paths, contributing to a similar closeness centrality value for many nodes.

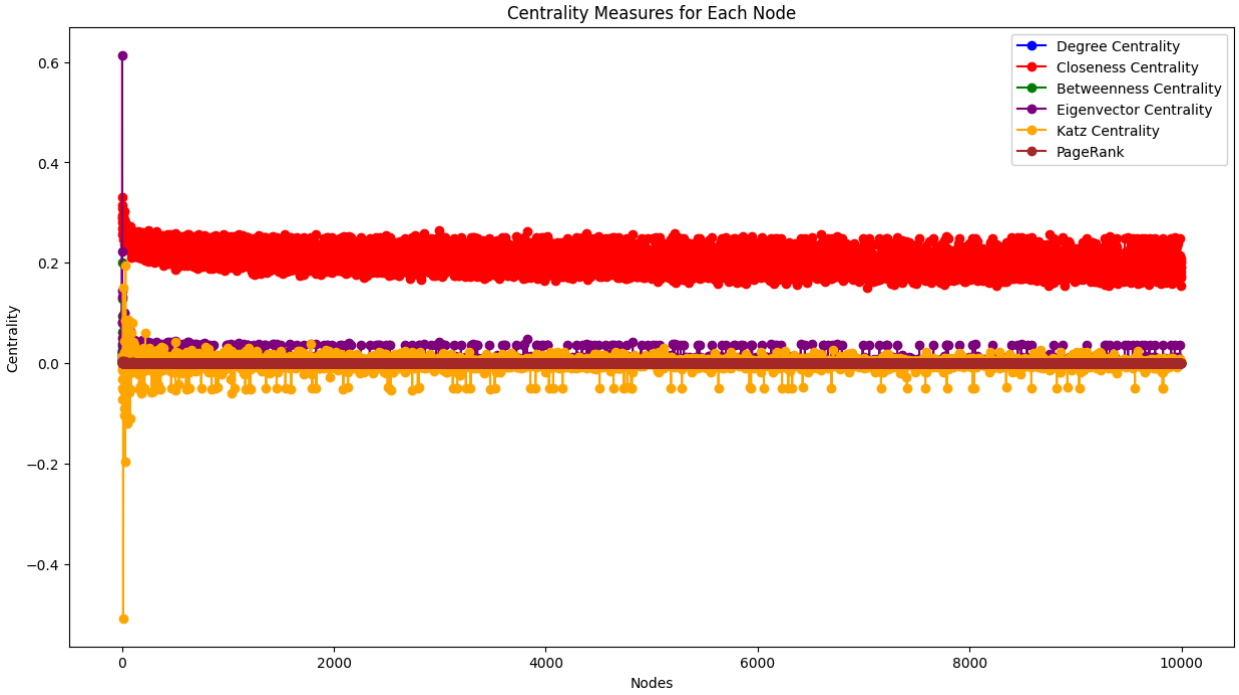
In summary, the nicely distributed closeness centrality graph for a Barabási–Albert network is a result of the network's scale-free degree distribution, which leads to the presence of hub nodes that are close to many other nodes, resulting in a relatively uniform closeness centrality distribution across nodes.

Top 10 Description

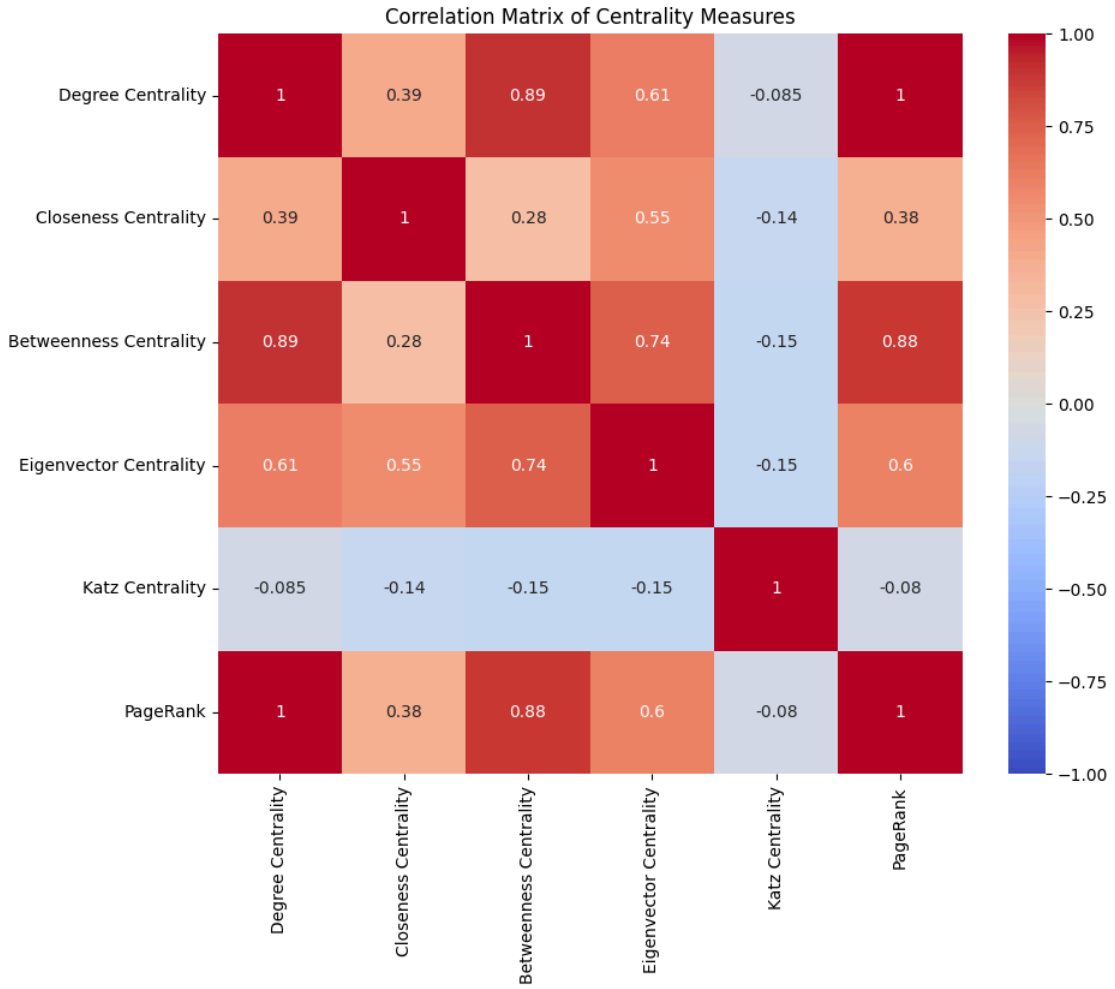
NODES	CLOSENESS CENTRALITY
3	0.330819
0	0.313950
5	0.308345
25	0.303727
1	0.295400
12	0.295400
8	0.291669
4	0.289490
27	0.286743
11	0.282561



Visualizing Centrality Measures Together



Correlation Matrix Between The Centrality Measures



In a Barabási–Albert (BA) network, high-degree nodes play a central role as hubs due to their numerous direct connections. These hubs also exhibit high betweenness centrality, as they often lie on many shortest paths between other nodes, facilitating efficient communication across the network. Additionally, nodes with high degrees tend to have high PageRank centrality, receiving incoming links from other vital nodes and further solidifying their central position in the network as other hubs frequently link to hubs.

The high correlation between these centrality measures in a BA network highlights the importance of nodes with high degrees. These nodes play a central role in direct connections, serve as bridges between different parts of the network, and are crucial for the network's overall structure and function. Their importance is reflected in their high centrality scores across

multiple measures, emphasizing their vital role in information dissemination, network resilience, and overall network dynamics.

The negative correlations between Katz centrality and other measures indicate that nodes with high Katz centrality play unique roles in the network's structure, potentially influencing distant parts of the network through indirect paths rather than being highly connected or influential in a local neighborhood.

Clustering Coefficients

Global Clustering Coefficient

```
#Calculate Global Clustering Coefficient
global_clustering_coefficient = nx.transitivity(S)

#Print Global Clustering Coefficient
print("Global Clustering Coefficient:", global_clustering_coefficient)
```

```
Global Clustering Coefficient: 0.0014043505610087 92
```

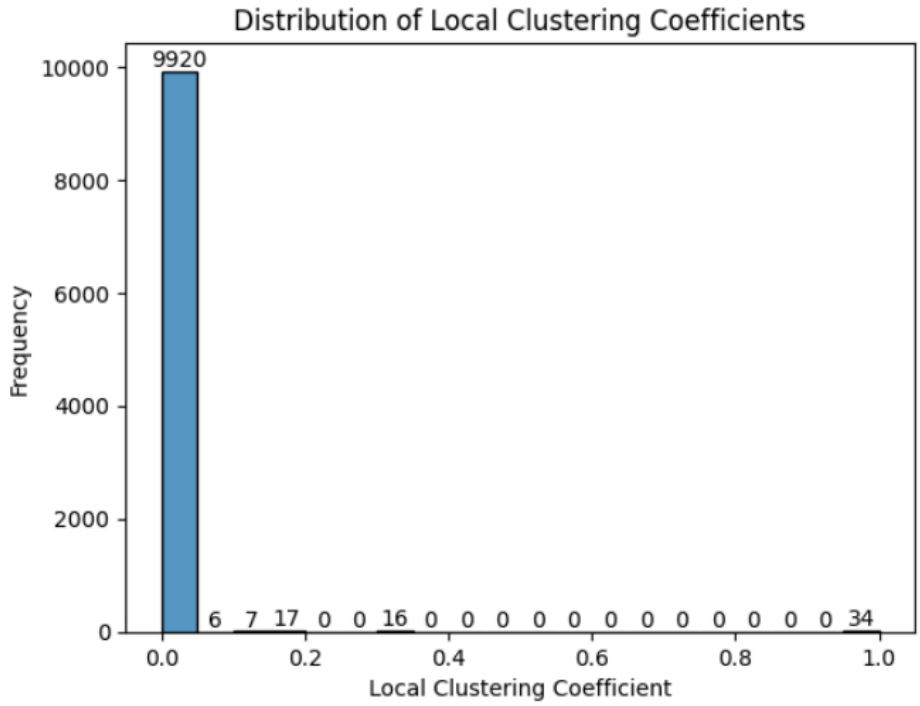
Local Clustering Coefficient

```
#Calculate Local Clustering Coefficient
local_clustering_coefficient = nx.clustering(G)

#Print each node and it's Clustering Coefficient
for node, local in local_clustering_coefficient.items():
    print(f"Node {node}: Local Clustering Coefficient = {local}")
```

```
top_10_nodes = sorted(local_clustering_coefficient.items(), key=lambda x: x[1], reverse=True)[:10]
nodes_with_coefficient_1 = sum(1 for coeff in local_clustering_coefficient.values() if coeff == 1)
print(f"Number of Nodes with Local Clustering Coefficient 1: {nodes_with_coefficient_1}")
```

```
Number of Nodes with Local Clustering Coefficient 1: 34
```

```python
#Create a line chart for the distribution of local clustering coefficients
coefficients = list(local_clustering_coefficient.values())
coefficients.sort()
unique_coefficients = list(set(coefficients))
counts = [coefficients.count(coeff) for coeff in unique_coefficients]

#Create a histogram for the distribution of local clustering coefficients
sns.histplot(list(local_clustering_coefficient.values()), bins=20, kde=False)
plt.xlabel('Local Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('Distribution of Local Clustering Coefficients')

#Display the frequency count on top of each bin
for p in plt.gca().patches:
    plt.gca().annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.get_height()),ha='center', va='bottom', color='black', fontsize=10)

plt.show()
```



Average Local Clustering Coefficient

```python
#Calculate Average Local Clustering Coefficient
average_local_clustering_coefficient = nx.average_clustering(S)

print("Average Local Clustering Coefficient:", average_local_clustering_coefficient)
```

```
Average Local Clustering Coefficient: 0.0044780905036862426
```

Transitivity and Reciprocity

Transitivity

```
transitivity = nx.transitivity(S)
print("Transitivity:", transitivity)
```

```
Transitivity: 0.0014043505610087 92
```

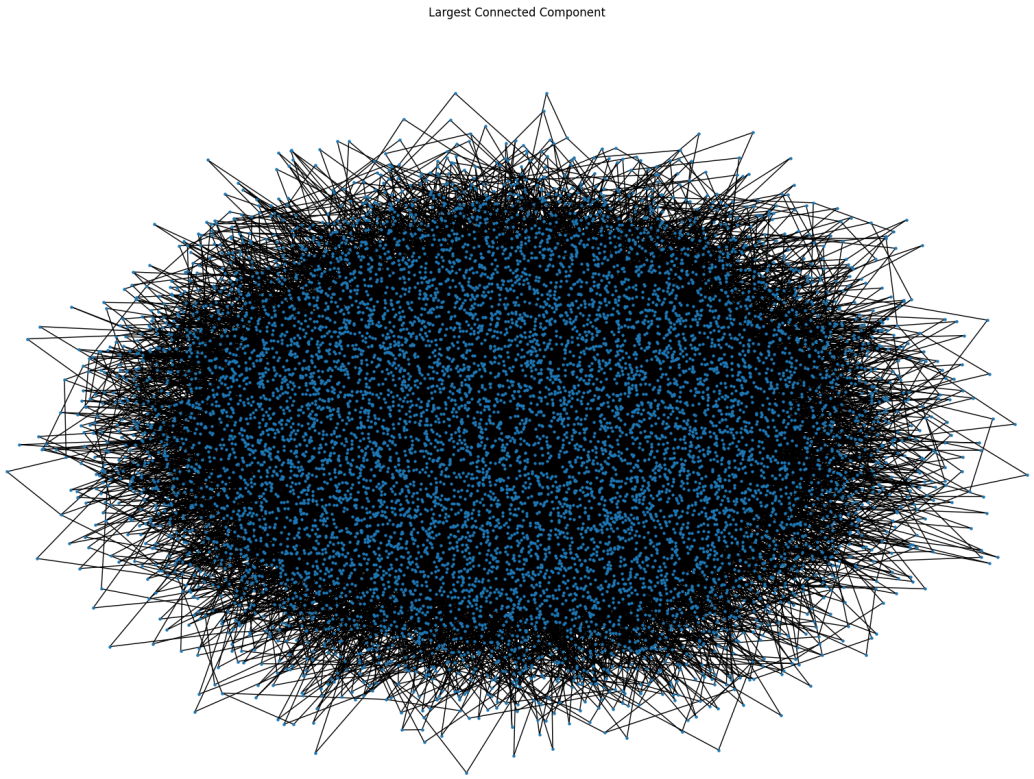
Reciprocity

Reciprocity is a simplified version of transitivity because it considers closed loops of length 2, which can only happen in directed graphs. However, in this case, our chosen network needs to be produced. Hence, we cannot find a value for the interchange of this network.

Largest Connected Component

```
#Find the largest connected component
largest_component = max(nx.connected_components(S), key=len)

#Create a subgraph of the largest connected component
largest_component_subgraph = S.subgraph(largest_component)

#Print the number of nodes and edges in the largest connected component
print("Number of nodes in the largest connected component:", largest_component_subgraph.number_of_nodes())
print("Number of edges in the largest connected component:", largest_component_subgraph.number_of_edges())

#Visualize the largest connected component
plt.figure(figsize=(18, 12))
nx.draw(largest_component_subgraph, with_labels=False, node_size = 5)
plt.title('Largest Connected Component')
plt.show()
```

```
Number of nodes in the largest connected component: 10000
Number of edges in the largest connected component: 19996
```



Comparing The Barabasi-Albert Network To Real Networks

Quantitative Comparison

Characteristic	Network 1 - Euroroads	Network 2 - General Relativity and Quantum Cosmology Collaboration Network	Barabasi-Albert Network
Number Of Nodes	1,174	5,242	10,000
Number Of Edges	1417	14,496	19,996
Max Degree	10 (Moscow)	81	264
Min Degree	1 (Greenock)	1	2
Average Degree	1.188	5.530	3.992
Standard Deviation Of Degree Distribution	2.414	7.918	6.534
Mean Degree Centrality	0.002057	0.001055	0.000399
Max Degree Centrality	0.008525 (Moscow)	0.015455	0.026403
Mean Eigenvector Centrality	0.001983	0.001614	0.002893
Max Eigenvector Centrality	0.365117 (Paris)	0.155561	0.613823
Mean Katz Centrality	0.000407	0.001991	0.002261
Max Katz Centrality	0.052258 (Moscow)	0.329264	0.193930
Mean PageRank Centrality	0.000852	0.000191	0.000100
Max PageRank Centrality	0.002783 (Moscow)	0.001428	0.005238
Mean Betweenness Centrality	0.011629	0.000606	0.000400

Max Betweenness Centrality	0.214794 (Brest)	0.037027	0.201154
Mean Closeness Centrality	0.045668	0.106301	0.201742
Max Closeness Centrality	0.076865 (Warsaw)	0.194285	0.330819
Global Clustering Coefficient	0.033886	0.629842	0.001404
Average Local Clustering Coefficient	0.016731	0.529635	0.004478
Transitivity	0.033886	0.629842	0.001404
Number Of Nodes In The Largest Connected Component	1,039	4,158	19,996
Number Of Edges In The Largest Connected Component	1,305	13,428	19,996

Qualitative Comparison

Characteristic	**Network 1 - Euroroads**	**Network 2 - General Relativity and Quantum Cosmology Collaboration Network**	**Barabasi-Albert Network**
Description	Real-world geographic network	Collaboration network in academia	Synthetic scale-free network
Size	Small to medium	Large	Large
Connectivity Patterns	Moderate clustering, transitivity	High clustering, transitivity	Low clustering, low transitivity
Centrality Measures	Lower centrality measures overall	Higher centrality measures	Moderate to high centrality measures
Degree Distribution	More uniform distribution	Heterogeneous degree distribution	Heterogeneous degree distribution

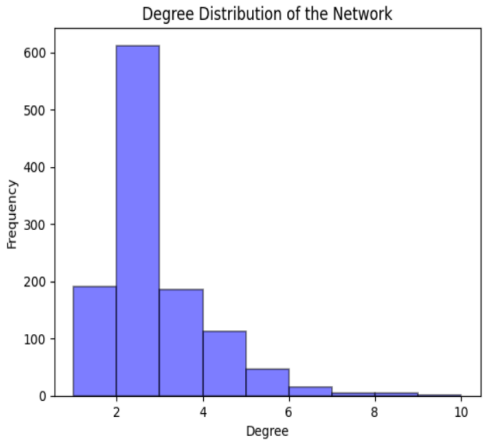
Growth Mechanism	Geographical constraints	Academic interests, cumulative advantage	Preferential attachment, cumulative advantage
Connected Components	Multiple connected components	Few connected components	Single connected component

Here, our author network shows some similarities to the Barabasi-Albert network, as some degree of preferential attachment also occurs in this network, similar to the Barabasi-Albert network. However, the Barabasi-Albert Model fails to capture additional real-world phenomena affecting the Author network and its growth.

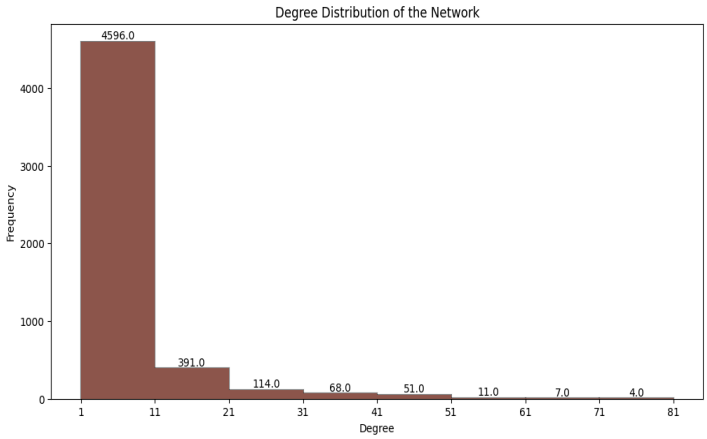
One prominent feature in both networks is the presence of hubs. Like how nodes in the Barabasi-Albert model prefer to attach themselves with nodes with a high degree, our Author network also displays this behaviour where people like collaborating with well-established researchers within the network. This preferential attachment results in the formation of hubs within this network as well.

In academia, there's also a phenomenon known as the "Matthew effect" or "cumulative advantage," where researchers with early success or recognition tend to receive more opportunities and collaborations, leading to further success. This principle is analogous to preferential attachment and can contribute to forming hubs in collaboration networks.

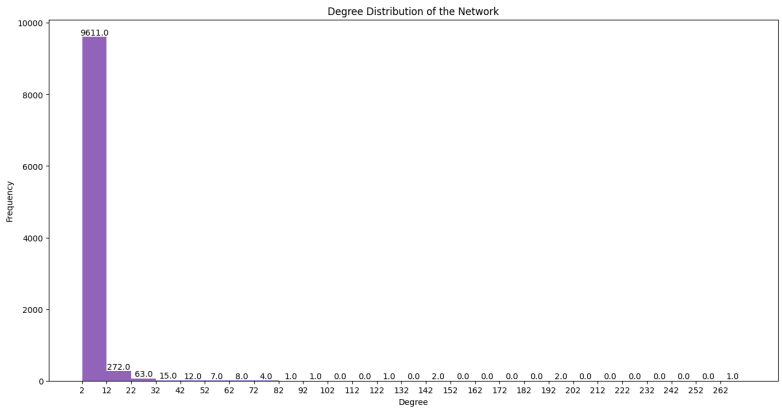
However, besides these similarities, certain real-world phenomena aren’t captured in the Barabasi-Albert Network. Researchers are also driven by academic interests, research collaborations, institutional affiliations, and the dynamics of scientific discovery within general relativity and quantum cosmology in our Author Network. Each researcher doesn't always prefer the most popular subfield of research. Moreover, the individual skills of researchers also play a huge role in how they associate in the real world. Moreover, besides research interests, interpersonal social dynamics also influence how people associate in the real world. Such interactions are where the behaviours of our network deviate from the Barabasi-Albert model.



Euroroads



General Relativity and Quantum Cosmology Collaboration



Barabasi-Albert Network

We have also presented the degree distributions for the three networks we have used for our study above. The heterogeneous nature of the degree distribution in the Barabasi-Albert and the Author collaboration networks is clearly evident above. The skewed distribution includes a few hubs with a high degree distribution, while most other nodes have a low degree. In contrast, the degree distribution for the Euroroads network is relatively uniform, as road networks don't expand based on preferential attachment; instead, they depend on geographical constraints such as terrain, population distribution, and economic development, influencing the layout and connectivity of roads within Europe.

Information Diffusion In The Networks

The Independent Cascade Model (ICM) is a fundamental concept in network theory used to simulate information diffusion in networks. In this model, each node in the network has an activation probability that determines the likelihood of it being influenced by its neighbours. The model proceeds in discrete steps, where at each step, activated nodes have a chance to activate their neighbours based on these probabilities.

Implementation Of The Independent Cascade Model (ICM)

The code implements the Independent Cascade Model on a given graph G. It begins by initializing activation probabilities for each node. Each node's total activation probability is divided among its neighbours, ensuring that the sum of probabilities for all neighbours is not greater than 1. This mechanism ensures a realistic diffusion process.

The *independent_cascade_model function* simulates the diffusion process starting from a set of initial nodes. It iterates through activated nodes at each step, attempting to activate their neighbours based on the activation probabilities. If a neighbour is successfully activated, it is added to the set of activated nodes for the next step. The function is called for a specific number of iterations (*num_iterations* = *10*). It randomly selects two start nodes at the beginning of each iteration, initializes activation probabilities, and runs the simulation. It then prints the start nodes, the set of activated nodes, and the number of steps taken for diffusion to stabilize (max_steps). Finally, the code calculates and prints the average number of steps required for diffusion to stabilize across all iterations.

```python
def initialize_activation_probabilities(G):
    for node in G.nodes():
        neighbors = list(G.neighbors(node))
        total_prob = 1.0
        G.nodes[node]['activation_probability'] = {}
        for neighbor in neighbors:
            p = random.uniform(0, total_prob)
            G.nodes[node]['activation_probability'][neighbor] = p
            total_prob -= p

def independent_cascade_model(G, initial_nodes):
    activated_nodes = set(initial_nodes)
    new_nodes_activated = True
    max_steps = 0

    while new_nodes_activated:
        new_nodes_activated = False
        new_activated_nodes = set()

        for node in activated_nodes:
            neighbors = list(G.neighbors(node))

            for neighbor in neighbors:
                if neighbor not in activated_nodes:
                    activation_probability = G.nodes[node]['activation_probability'][neighbor]
                    if random.random() < activation_probability:
                        new_activated_nodes.add(neighbor)
                        new_nodes_activated = True

        activated_nodes |= new_activated_nodes
        max_steps += 1

    return len(activated_nodes), max_steps
```

Applying ICM On Our Chosen Networks

Network 1 - Euroroads

```python
for _ in range(num_iterations):
    start_nodes = random.sample(list(G.nodes()), 2)
    initial_nodes = start_nodes
    initialize_activation_probabilities(G)
    activated_nodes, max_steps = independent_cascade_model(G, initial_nodes)
    total_steps += max_steps
    print("Start Nodes:", start_nodes)
    print("Activated Nodes:", activated_nodes)
    print("Max Steps:", max_steps)
    print("-------------------------")

average_steps = total_steps / num_iterations
print("Average number of steps required:", average_steps)
```

```
Start Nodes: ['Zhaksy', 'Abrantes']
Activated Nodes: 5
Max Steps: 3
-------------------------
Start Nodes: ['Agrinio', 'Warsaw']
Activated Nodes: 569
Max Steps: 101
-------------------------
Start Nodes: ['Oradea', 'Torres Novas']
Activated Nodes: 599
Max Steps: 84
-------------------------
Start Nodes: ['Rovaniemi', 'Korczowa']
Activated Nodes: 697
Max Steps: 115
-------------------------
Start Nodes: ['Braunschweig', 'Manisa']
Activated Nodes: 625
Max Steps: 98
-------------------------
Start Nodes: ['Sarreguemines', 'Kiev']
Activated Nodes: 4
Max Steps: 2
Start Nodes: ['Sarreguemines', 'Kiev']
Activated Nodes: 4
Max Steps: 2
-------------------------
Start Nodes: ['Kaunas', 'Pristina']
Activated Nodes: 7
Max Steps: 4
-------------------------
Start Nodes: ['Sary Tash', 'Avellino']
Activated Nodes: 11
Max Steps: 5
-------------------------
Start Nodes: ['Valmiera', 'Örnsköldsvik']
Activated Nodes: 343
Max Steps: 65
-------------------------
Start Nodes: ['Astara', 'Yekaterinburg']
Activated Nodes: 26
Max Steps: 12
-------------------------
Average number of steps required: 48.9
```

Network 2 - General Relativity And Quantum Cosmology Collaboration

```python
for _ in range(num_iterations):
    start_nodes = random.sample(list(G.nodes()), 2)
    initial_nodes = start_nodes
    initialize_activation_probabilities(G)
    activated_nodes, max_steps = independent_cascade_model(G, initial_nodes)
    total_steps += max_steps
    print("Start Nodes:", start_nodes)
    print("Activated Nodes:", activated_nodes)
    print("Max Steps:", max_steps)
    print("-------------------------")

average_steps = total_steps / num_iterations
print("Average number of steps required:", average_steps)
```

```
Start Nodes: ['15962', '12788']
Activated Nodes: 2570
Max Steps: 205
-------------------------
Start Nodes: ['19442', '1910']
Activated Nodes: 12
Max Steps: 6
-------------------------
Start Nodes: ['14009', '9915']
Activated Nodes: 2493
Max Steps: 201
-------------------------
Start Nodes: ['24340', '18648']
Activated Nodes: 2370
Max Steps: 179
-------------------------
Start Nodes: ['2410', '1254']
Activated Nodes: 2
Max Steps: 1
-------------------------
Start Nodes: ['14638', '24942']
Activated Nodes: 2523
Max Steps: 192
```

```
Start Nodes: ['14638', '24942']
Activated Nodes: 2523
Max Steps: 192
-------------------------
Start Nodes: ['16765', '17992']
Activated Nodes: 2601
Max Steps: 241
-------------------------
Start Nodes: ['11114', '22605']
Activated Nodes: 3
Max Steps: 2
-------------------------
Start Nodes: ['21861', '19510']
Activated Nodes: 9
Max Steps: 5
-------------------------
Start Nodes: ['23248', '20435']
Activated Nodes: 2307
Max Steps: 139
-------------------------
Average number of steps required: 234.6
```

Network 3 - Barabasi-Albert Network

```
num_iterations = 10
total_steps = 0
for _ in range(num_iterations):
    start_nodes = random.sample(list(S.nodes()), 2)
    initial_nodes = start_nodes
    initialize_activation_probabilities(S)
    activated_nodes, max_steps = independent_cascade_model(S, initial_nodes)
    total_steps += max_steps
    print("Start Nodes:", start_nodes)
    print("Activated Nodes:", activated_nodes)
    print("Max Steps:", max_steps)
    print("-------------------------")

average_steps = total_steps / num_iterations
print("Average number of steps required:", average_steps)
```

```
Start Nodes: [5536, 5088]
Activated Nodes: 7898
Max Steps: 263
-------------------------
Start Nodes: [7110, 6045]
Activated Nodes: 7983
Max Steps: 280
-------------------------
Start Nodes: [2694, 8025]
Activated Nodes: 8127
Max Steps: 302
-------------------------
Start Nodes: [3013, 2053]
Activated Nodes: 8087
Max Steps: 304
-------------------------
Start Nodes: [5809, 4941]
Activated Nodes: 7686
Max Steps: 243
-------------------------
Start Nodes: [4458, 5174]
Activated Nodes: 7946
Max Steps: 258
-------------------------
Start Nodes: [4458, 5174]
Activated Nodes: 7946
Max Steps: 258
-------------------------
Start Nodes: [3783, 9618]
Activated Nodes: 7549
Max Steps: 196
-------------------------
Start Nodes: [9952, 2943]
Activated Nodes: 8126
Max Steps: 331
-------------------------
Start Nodes: [9409, 6734]
Activated Nodes: 8029
Max Steps: 298
-------------------------
Start Nodes: [8224, 2055]
Activated Nodes: 7958
Max Steps: 288
-------------------------
Average number of steps required: 276.3
```

In the Euroroads Network, the connectivity patterns exhibit moderate clustering and transitivity. This means that while there is some clustering of nodes and local interconnectedness, portions of the network are not tightly connected to the main component. Specifically, there are approximately 150 nodes that are not part of the largest connected component.
When selecting two random nodes for the diffusion process, there is a possibility that both selected nodes are not part of the largest connected component. This can interfere with the diffusion process in several ways:

1. **Limited Influence:** Nodes outside the largest connected component have limited influence on the rest of the network. If both randomly selected nodes are not part of the main component, they may have fewer connections to other nodes, reducing their ability to activate additional nodes.
2. **Isolation:** Isolated nodes or nodes in smaller components may not receive information from the main component efficiently. This can lead to slower diffusion or a smaller overall number of activated nodes.

3. **Fragmentation:** Multiple connected components can lead to fragmentation of the diffusion process. Activated nodes in smaller components may not be able to influence nodes in other components effectively, leading to disjointed activation patterns.
4. **Limited Reach:** Nodes outside the largest component may have a limited reach for activating other nodes. This can result in fewer nodes being activated overall, especially if the randomly selected nodes are not well-connected to the rest of the network.
5. **Stalled Diffusion:** If the randomly selected nodes are not part of the largest component and do not have sufficient connections to activate other nodes, the diffusion process may stall or proceed very slowly, leading to fewer activated nodes than a more connected network.

In the General Relativity and Quantum Cosmology Collaboration Network and the Barabasi-Albert Network, the presence of hubs and greater overall connectivity significantly impacts the diffusion process compared to the Euroroads Network.

1. **Hubs and Connectivity:** Both networks have hubs and higher overall connectivity.
2. **Impact on Diffusion:** Hubs activate many nodes, leading to more activation and a longer diffusion process.
3. **Number of Activated Nodes:** These networks have more activated nodes.
4. **Steps to Maximum Activation:** Due to hubs and connectivity, it takes more steps to reach maximum activation.
5. **Influential Nodes:** Hubs are crucial in activating a large portion of the network.

Overall, the presence of hubs and greater connectivity in the General Relativity and Quantum Cosmology Collaboration Network and the Barabasi-Albert Network leads to a higher number of activated nodes and a more prolonged diffusion process compared to the Euroroads Network, where connectivity is lower, and nodes are less likely to be part of a large connected component.

If you wish to check the code for the project, refer to the following GitHub link:
https://github.com/RonitGupta2002/SNA-Project-Network-Analysis