ROUND 1

1 Dataset 1: College Message Network

1.1 Dataset Description

This dataset represents private message exchanges between users in an online social network at the University of California, Irvine. It captures user interactions over time, showing who communicated with whom and when. The dataset is structured as a temporal directed network, meaning each interaction has a direction (sender $\rightarrow$ receiver) and a timestamp.

The dataset is designed for social network analysis and has been formatted for use in SNAP (Stanford Network Analysis Platform), a widely used framework for analyzing large-scale networks.

1.1.1 Nodes (Users)

- Each node represents a unique user in the online social network.
- Users could search for others on the platform and initiate private conversations.
- Since this is a social network, users likely had profiles, and interactions could be based on shared interests.

1.1.2 Edges (Messages)

- Each edge (u, v, t) represents a directed interaction, meaning: user u sent a message to user v at time t
- Since the edges are directed, the network is asymmetric, meaning: If u sends a message to v, it does not imply v responded
- The presence of timestamps makes this a temporal network, enabling the study of interaction patterns over time

1.1.3 Data Format

- `FromNodeId` – ID of the sender (user who sent the message)
- `ToNodeId` – ID of the receiver (user who received the message)

1.1.4 Dataset Statistics

```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Load the dataset
file_path = "CollegeMsg.txt"
G = nx.Graph()

# Count temporal edges
temporal_edges_count = 0

with open(file_path, 'r') as file:
    for line in file:
        u, v, t = map(int, line.strip().split())
        G.add_edge(u, v)
        temporal_edges_count += 1

# Dataset statistics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()

print("Dataset Statistics:")
print(f"Nodes: {num_nodes}")
print(f"Temporal Edges: {temporal_edges_count}")
print(f"Edges in static graph: {num_edges}")
print(f"Time span: 193 days")
```

Metric	Value
Total Nodes (Users)	1,899
Total Temporal Edges (Messages)	59,835
Edges in Static Graph	20,296
Time Span	193 days

- **Nodes (1,899):** There are 1,899 distinct users who participated in at least one conversation

- **Temporal Edges (59,835):** The dataset records 59,835 unique private messages over the 193-day period
- **Time Span (193 days):** The dataset covers roughly 6 months, meaning we can analyze:
  - How messaging patterns evolve
  - When users are most active
  - How long conversations last

1.2 Data Characteristics

- **Directed Graph**
  - The network is not symmetric
  - If A sends a message to B, it does not mean B responded
- **Temporal Structure**
  - Since each message has a timestamp, we can analyze activity trends
  - We can track peak usage times and response patterns
- **Repeated Interactions**
  - If multiple edges exist between the same pair of nodes, we can analyze conversational intensity
- **Sparse Graph**
  - With 1,899 nodes but only 20,296 unique edges (ignoring timestamps), the network is not fully connected
  - Some users may have sent messages to only a few others

1.3 Potential Use Cases

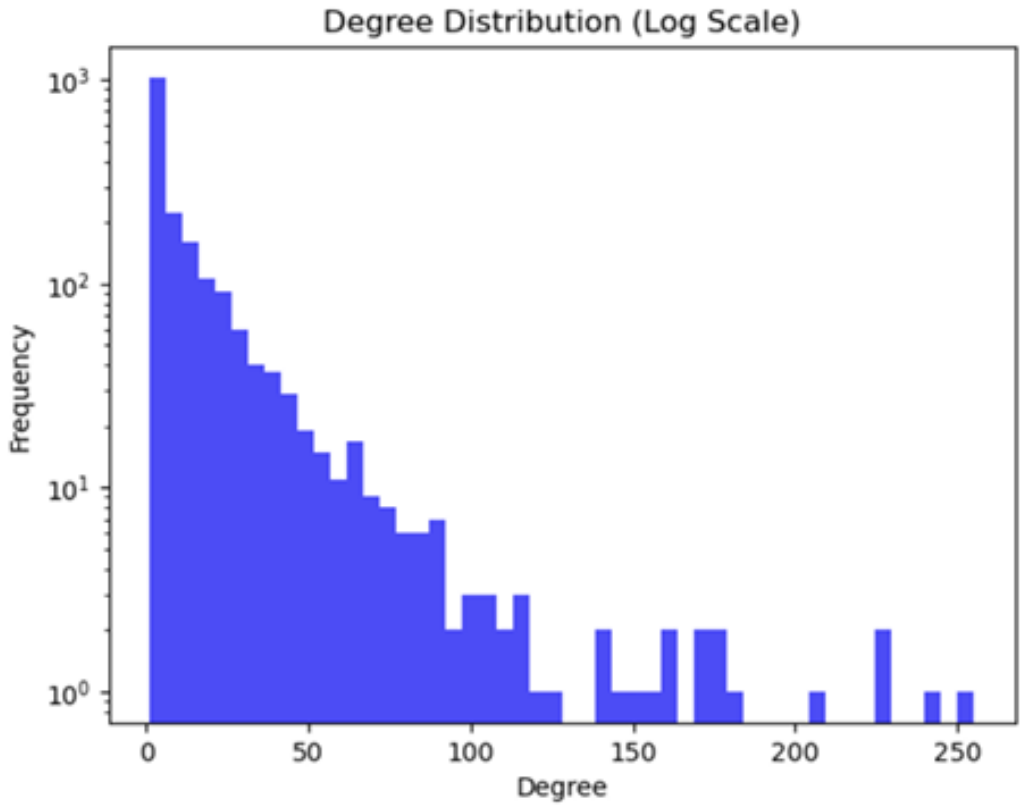
- **Social Network Analysis**
  - Identifying influential users based on message activity
  - Finding message patterns (who interacts with whom)
- **Community Detection**
  - Identifying closely connected user groups
  - Detecting friend circles or communication clusters
- **Temporal Analysis**
  - Understanding how user activity changes over days, weeks, or months
  - Analyzing response times (how quickly users reply to messages)

- **Network Evolution**
  - Studying user engagement trends over time
  - Predicting whether new users will engage based on past behavior

2 Network Statistics

2.0.1 Degree Distribution Analysis

```python
# Calculate degree distribution
degrees = [G.degree(n) for n in G.nodes()]
max_degree = max(degrees)
min_degree = min(degrees)
avg_degree = np.mean(degrees)
std_degree = np.std(degrees)

# Plot degree distribution in log scale
plt.hist(degrees, bins=50, log=True, color='blue', alpha=0.7)
plt.title("Degree Distribution (Log Scale)")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.show()

# Print degree statistics
print("\nDegree Statistics:")
print(f"Max Degree: {max_degree}")
print(f"Min Degree: {min_degree}")
print(f"Average Degree: {avg_degree:.2f}")
print(f"Standard Deviation of Degree Distribution: {std_degree:.2f}"
    )
```




```
Degree Statistics:
Max Degree: 255
Min Degree: 1
Average Degree: 14.57
Standard Deviation of Degree Distribution: 24.46
```

Figure 1: Network graph visualization showing message patterns between users

3 Centrality Measures

```python
# Compute centrality measures
degree_centrality = nx.degree_centrality(G)
eigenvector_centrality = nx.eigenvector_centrality(G, max_iter=1000)
katz_centrality = nx.katz_centrality(G, alpha=0.01, beta=1.0,
   max_iter=1000)
pagerank_centrality = nx.pagerank(G, alpha=0.85)
betweenness_centrality = nx.betweenness_centrality(G)
closeness_centrality = nx.closeness_centrality(G)

# Clustering coefficients
```

```
local_clustering_coefficients = nx.clustering(G)
global_clustering_coefficient = nx.average_clustering(G)
avg_local_clustering_coefficient = np.mean(list(
    local_clustering_coefficients.values()))

# Function to get top 10 nodes for a centrality measure
def get_top_10(centrality_dict, measure_name):
    sorted_centrality = sorted(centrality_dict.items(), key=lambda x
        : x[1], reverse=True)
    top_10 = sorted_centrality[:10]
    return pd.DataFrame(top_10, columns=["Node", measure_name])

# Get top 10 nodes for each centrality measure
top_10_degree = get_top_10(degree_centrality, "Degree Centrality")
top_10_eigenvector = get_top_10(eigenvector_centrality, "Eigenvector
    Centrality")
top_10_katz = get_top_10(katz_centrality, "Katz Centrality")
top_10_pagerank = get_top_10(pagerank_centrality, "PageRank
    Centrality")
top_10_betweenness = get_top_10(betweenness_centrality, "Betweenness
    Centrality")
top_10_closeness = get_top_10(closeness_centrality, "Closeness
    Centrality")
top_10_local_clustering = get_top_10(local_clustering_coefficients,
    "Local Clustering Coefficient")

# Display top 10 tables
print("Top 10 Nodes for Degree Centrality:")
print(top_10_degree)

print("\nTop 10 Nodes for Eigenvector Centrality:")
print(top_10_eigenvector)

print("\nTop 10 Nodes for Katz Centrality:")
print(top_10_katz)

print("\nTop 10 Nodes for PageRank Centrality:")
print(top_10_pagerank)
```

```
41
42 print("\nTop 10 Nodes for Betweenness Centrality:")
43 print(top_10_betweenness)
44
45 print("\nTop 10 Nodes for Closeness Centrality:")
46 print(top_10_closeness)
47
48 print("\nTop 10 Nodes for Local Clustering Coefficient:")
49 print(top_10_local_clustering)
```

3.1 Top 10 Nodes by Centrality Measures

3.1.1 Degree Centrality

Rank	Node	Degree Centrality
1	103	0.134352
2	9	0.126976
3	105	0.119600
4	400	0.119600
5	32	0.109062
6	41	0.096417
7	3	0.093783
8	42	0.093256
9	249	0.090622
10	638	0.089041

Table 1: Top 10 Nodes by Degree Centrality

3.1.2 Eigenvector Centrality

Rank	Node	Eigenvector Centrality
1	103	0.156268
2	105	0.156126
3	32	0.150791
4	9	0.142565
5	249	0.122914
6	638	0.122764
7	400	0.121178
8	3	0.121055
9	372	0.116349
10	194	0.116088

Table 2: Top 10 Nodes by Eigenvector Centrality

3.1.3 Katz Centrality

Rank	Node	Katz Centrality
1	103	0.086568
2	105	0.082902
3	9	0.082192
4	32	0.078711
5	400	0.075915
6	3	0.068822
7	249	0.067828
8	41	0.067799
9	638	0.067518
10	42	0.065832

Table 3: Top 10 Nodes by Katz Centrality

3.1.4 PageRank Centrality

Rank	Node	PageRank Centrality
1	9	0.008822
2	400	0.008529
3	103	0.008017
4	105	0.007727
5	32	0.007019
6	42	0.006533
7	41	0.006142
8	3	0.006076
9	249	0.005436
10	713	0.005277

Table 4: Top 10 Nodes by PageRank Centrality

3.1.5 Betweenness Centrality

Rank	Node	Betweenness Centrality
1	9	0.064649
2	400	0.059931
3	105	0.059218
4	32	0.052021
5	103	0.048104
6	42	0.044027
7	3	0.040971
8	41	0.033643
9	523	0.031161
10	249	0.030816

Table 5: Top 10 Nodes by Betweenness Centrality

3.1.6 Closeness Centrality

Rank	Node	Closeness Centrality
1	32	0.476508
2	105	0.475907
3	9	0.461807
4	3	0.460791
5	638	0.450339
6	103	0.448732
7	42	0.448625
8	249	0.448198
9	598	0.447986
10	400	0.445131

Table 6: Top 10 Nodes by Closeness Centrality

3.2 Clustering Analysis

3.2.1 Top Nodes by Local Clustering Coefficient

Rank	Node	Local Clustering Coefficient
1	17	1.0
2	89	1.0
3	117	1.0
4	179	1.0
5	292	1.0
6	383	1.0
7	406	1.0
8	417	1.0
9	435	1.0
10	461	1.0

Table 7: Top 10 Nodes by Local Clustering Coefficient

```
# Check if the graph is directed
if nx.is_directed(G):
    print("The graph is directed.")
    reciprocity = nx.reciprocity(G)
else:
```

```
    print("The graph is undirected.")
    reciprocity = 1.0  # Undirected graphs have reciprocity = 1.0

# Transitivity
transitivity = nx.transitivity(G)

# Print global and average local clustering coefficients
print(f"\nGlobal Clustering Coefficient: {global_clustering_coefficient}")
print(f"Average Local Clustering Coefficient: {avg_local_clustering_coefficient}")

# Print results
print(f"Reciprocity: {reciprocity}")
print(f"Transitivity: {transitivity}")
```

3.2.2 Clustering Coefficient Comparison

- Global Clustering Coefficient: 0.109398
- Average Local Clustering Coefficient: 0.109398

3.2.3 Reciprocity and Transitivity

- The graph is undirected
- Reciprocity: 1.0
- Transitivity: 0.0568

3.3 Appropriate Centrality Measures for College Message Network

- **Selected Measures:** PageRank, Betweenness, Closeness
- **Less Preferred Measures:** Degree, Eigenvector, Katz, Clustering

3.3.1 Visualization of Centrality Measures

```
# Visualize centrality distributions
plt.figure(figsize=(18, 12))

# Degree Centrality
plt.subplot(2, 3, 1)
plt.hist(list(degree_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Degree Centrality Distribution")
plt.xlabel("Degree Centrality")
plt.ylabel("Frequency")

# Eigenvector Centrality
plt.subplot(2, 3, 2)
plt.hist(list(eigenvector_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Eigenvector Centrality Distribution")
plt.xlabel("Eigenvector Centrality")
plt.ylabel("Frequency")

# Katz Centrality
plt.subplot(2, 3, 3)
plt.hist(list(katz_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Katz Centrality Distribution")
plt.xlabel("Katz Centrality")
plt.ylabel("Frequency")

# PageRank Centrality
plt.subplot(2, 3, 4)
plt.hist(list(pagerank_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("PageRank Centrality Distribution")
plt.xlabel("PageRank Centrality")
plt.ylabel("Frequency")

# Betweenness Centrality
```

```
plt.subplot(2, 3, 5)
plt.hist(list(betweenness_centrality.values()), bins=50, log=True,
    color='blue', alpha=0.7)
plt.title("Betweenness Centrality Distribution")
plt.xlabel("Betweenness Centrality")
plt.ylabel("Frequency")

# Closeness Centrality
plt.subplot(2, 3, 6)
plt.hist(list(closeness_centrality.values()), bins=50, log=True,
    color='blue', alpha=0.7)
plt.title("Closeness Centrality Distribution")
plt.xlabel("Closeness Centrality")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

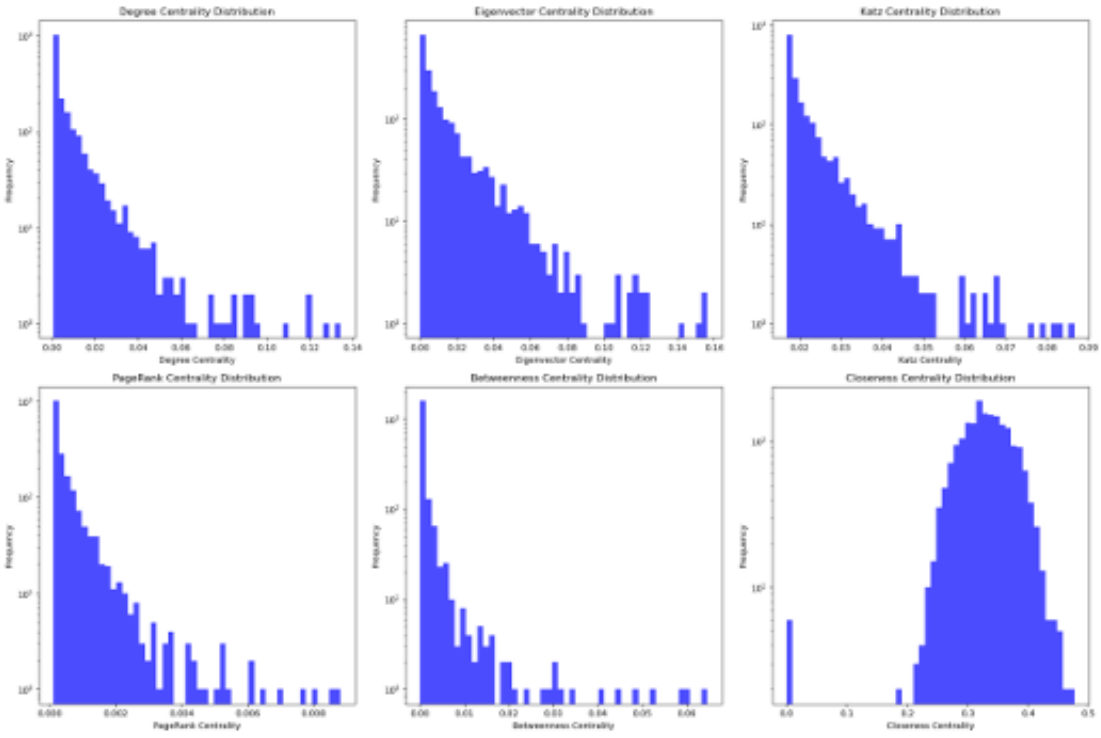


Figure 2: Distribution plots for six centrality measures showing node importance patterns

3.3.2 Visualization of Clustering Coefficients

```
# Create a figure with 2 rows and 1 column
plt.figure(figsize=(10, 12))

# First subplot: Distribution of local clustering coefficients
plt.subplot(2, 1, 1)
plt.hist(list(local_clustering_coefficients.values()), bins=50,
    color='blue', alpha=0.7)
plt.title("Distribution of Local Clustering Coefficients")
plt.xlabel("Local Clustering Coefficient")
plt.ylabel("Frequency")

# Second subplot: Distribution with global clustering coefficient
plt.subplot(2, 1, 2)
plt.hist(list(local_clustering_coefficients.values()), bins=50,
    color='blue', alpha=0.7,
         label="Local Clustering Coefficients")
plt.axvline(global_clustering_coefficient, color='red', linestyle='
    dashed', linewidth=2,
            label=f"Global Clustering Coefficient: {
               global_clustering_coefficient:.4f}")
plt.title("Comparison of Local and Global Clustering Coefficients")
plt.xlabel("Clustering Coefficient")
plt.ylabel("Frequency")
plt.legend()

plt.tight_layout()
plt.show()
```

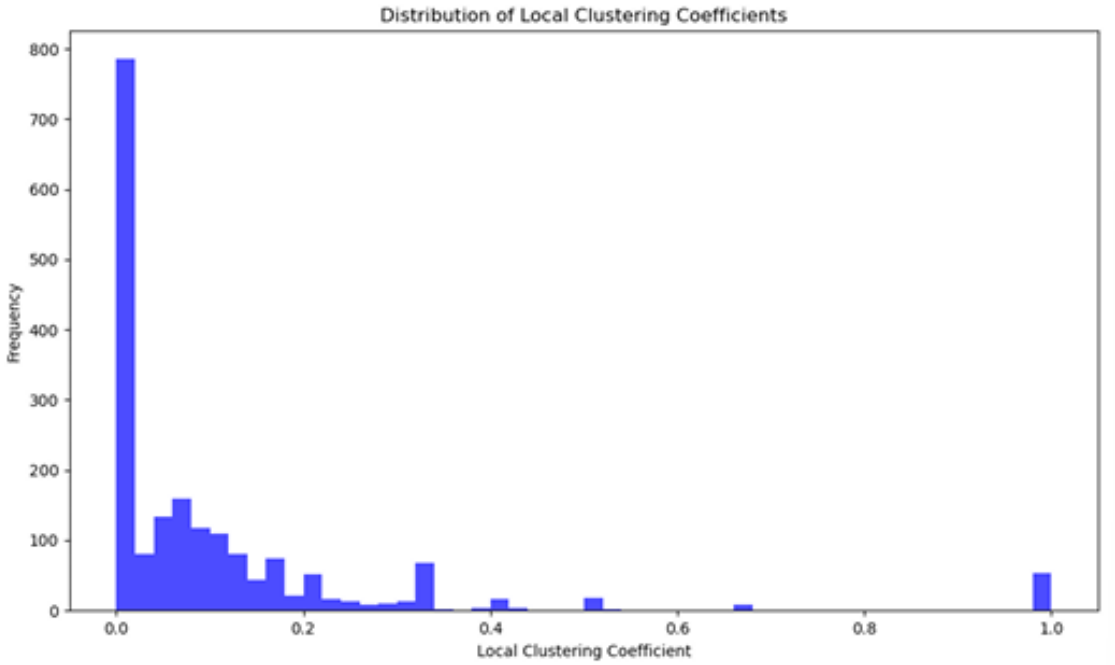


Figure 3: Distribution of local clustering coefficients across nodes

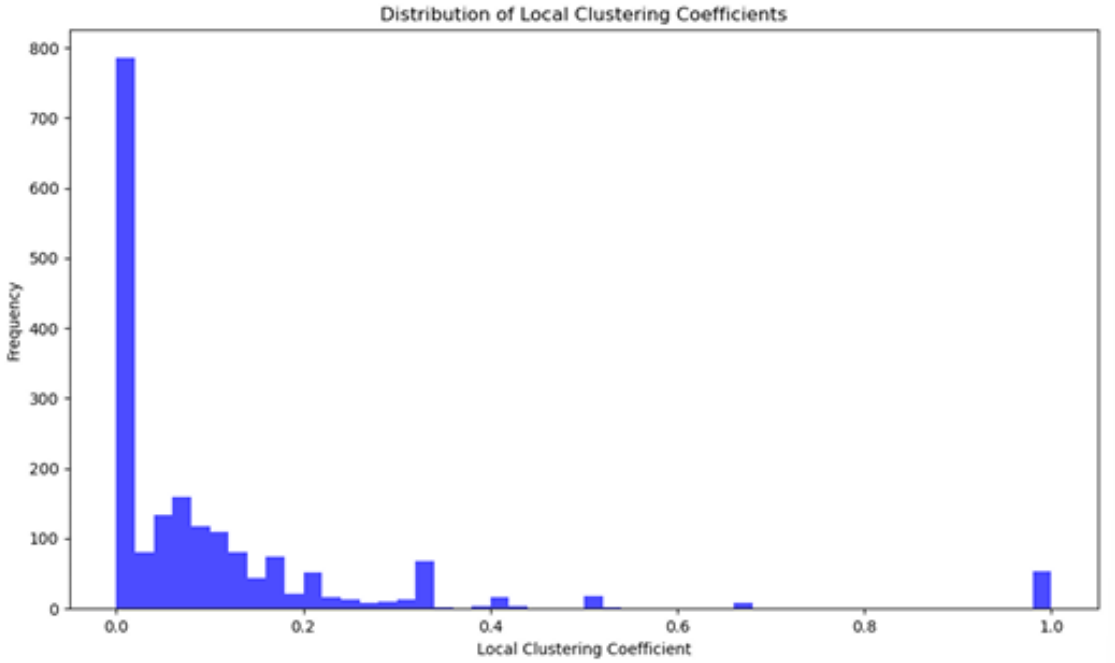


Figure 4: Comparison between local clustering coefficients and global clustering coefficient

4 Dataset 2: Condensed Matter Collaboration Network

4.1 Dataset Description

This dataset represents scientific collaborations between authors who have co-authored research papers in the Condensed Matter (COND-MAT) category on arXiv. The dataset spans from January 1993 to April 2003 (a period of 124 months) and captures how authors collaborate over time. It is structured as an undirected network, meaning that an edge between two authors indicates mutual collaboration.

4.1.1 Nodes and Edges

1. **Nodes (Authors)**
   - Each node represents an author who has published at least one paper in the Condensed Matter section of arXiv
2. **Edges (Collaborations)**
   - If two authors i and j co-authored a paper, an undirected edge connects them
   - If a paper has k authors, it forms a fully connected subgraph (a clique) with $\frac{k(k-1)}{2}$ edges
3. **Data Format**
   - `FromNodeId` – ID of an author
   - `ToNodeId` – ID of an author
   - The presence of an edge means these two authors co-authored at least one paper together

4.1.2 Data Statistics

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
file_path = "CA-CondMat.txt"
G = nx.read_edgelist(file_path, nodetype=int)
```

```python

# Basic network statistics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
print(f"Number of nodes: {num_nodes}")
print(f"Number of edges: {num_edges}")

# Largest connected component
largest_cc = max(nx.connected_components(G), key=len)
subgraph = G.subgraph(largest_cc)
print(f"Number of nodes in largest connected component: {subgraph.number_of_nodes()}")
print(f"Number of edges in largest connected component: {subgraph.number_of_edges()}")

# Average clustering coefficient
avg_clustering_coeff = nx.average_clustering(G)
print(f"Average clustering coefficient: {avg_clustering_coeff:.4f}")

# Number of triangles
triangles = sum(nx.triangles(G).values()) // 3
print(f"Number of triangles: {triangles}")
```

Property	**Value**
Total Nodes (Authors)	23,133
Total Edges (Collaborations)	93,497
Nodes in Largest WCC	21,363 (92.3%)
Edges in Largest WCC	91,342 (97.7%)
Nodes in Largest SCC	21,363 (92.3%)
Edges in Largest SCC	91,342 (97.7%)
Average Clustering Coefficient	0.6334
Number of Triangles	173,361
Fraction of Closed Triangles	0.107
Diameter (Longest Shortest Path)	14
90% Effective Diameter	6.5

Table 8: Statistical properties of the Condensed Matter Collaboration Network

4.2 Data Characteristics

1. **Undirected Graph**
   - Collaboration is mutual
   - If A co-authored a paper with B, both are connected
2. **High Clustering**
   - The average clustering coefficient (0.6334) shows authors tend to collaborate in tight groups
3. **Small-World Property**
   - The diameter of 14 means any two authors are connected by at most 14 steps
   - The 90% effective diameter of 6.5 indicates most authors are much closer

4.3 Potential Use Cases

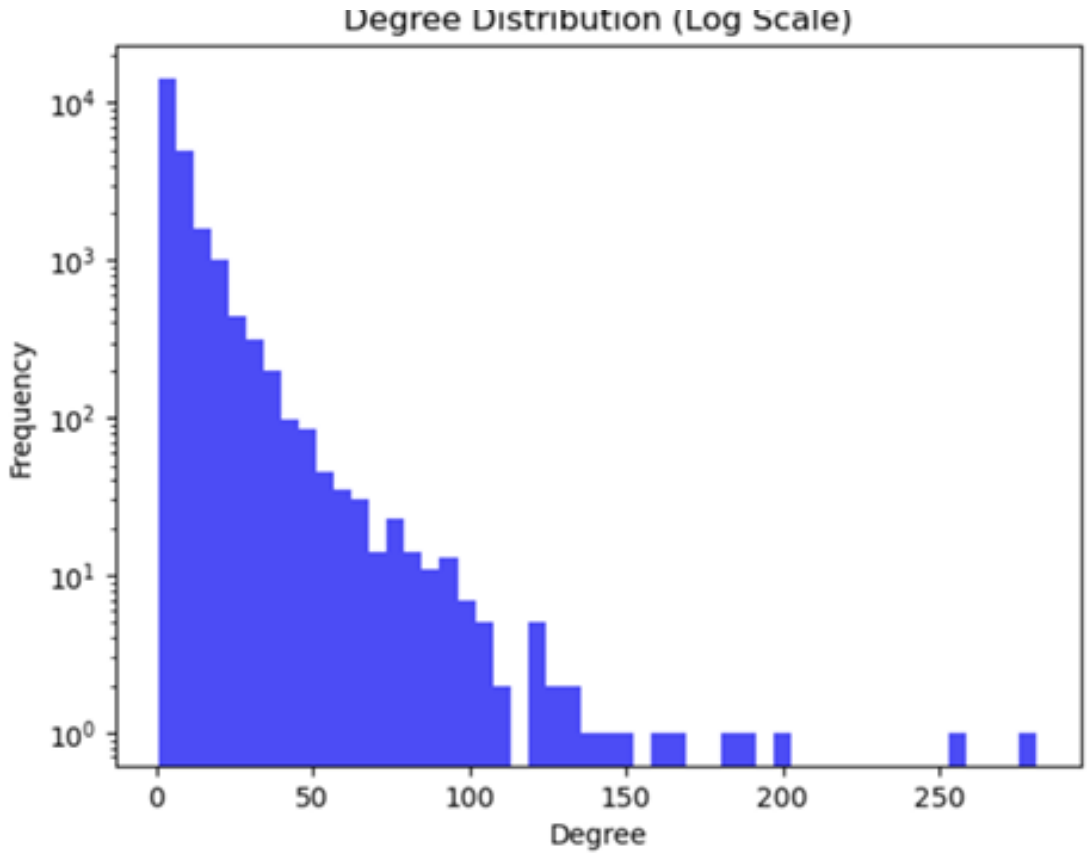
1. **Collaboration Network Analysis**
   - Identifying influential researchers
   - Finding central figures in scientific communities
2. **Community Detection**
   - Discovering research clusters (groups that frequently work together)
3. **Co-authorship Prediction**
   - Predicting future collaborations based on network patterns
4. **Network Evolution**
   - Analyzing how scientific collaborations grow and change over time

5 Network Statistics

5.0.1 Degree Distribution Analysis

```
# Degree distribution
degrees = [degree for node, degree in G.degree()]
max_degree = max(degrees)
min_degree = min(degrees)
avg_degree = np.mean(degrees)
std_degree = np.std(degrees)

```

```
# Plot degree distribution
plt.hist(degrees, bins=50, log=True, color='blue', alpha=0.7)
plt.title("Degree Distribution (Log Scale)")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.show()

print(f"Maximum degree: {max_degree}")
print(f"Minimum degree: {min_degree}")
print(f"Average degree: {avg_degree:.2f}")
print(f"Standard deviation of degree distribution: {std_degree:.2f}"
    )
```




```
Maximum degree: 281
Minimum degree: 1
Average degree: 8.083430596982666
Standard deviation of degree distribution: 10.645189586789236
```

Figure 5: Condensed Matter collaboration network visualization

6 Centrality Measures

```
# Compute centrality measures
degree_centrality = nx.degree_centrality(G)
eigenvector_centrality = nx.eigenvector_centrality(G, max_iter=1000)
katz_centrality = nx.katz_centrality(G, alpha=0.01, beta=1.0,
    max_iter=1000)
pagerank_centrality = nx.pagerank(G, alpha=0.85)
betweenness_centrality = nx.betweenness_centrality(G)
closeness_centrality = nx.closeness_centrality(G)

# Clustering coefficients
local_clustering_coefficients = nx.clustering(G)
global_clustering_coefficient = nx.average_clustering(G)
avg_local_clustering_coefficient = np.mean(list(
    local_clustering_coefficients.values()))

# Function to get top 10 nodes for a centrality measure
def get_top_10(centrality_dict, measure_name):
    sorted_centrality = sorted(centrality_dict.items(), key=lambda x
        : x[1], reverse=True)
    top_10 = sorted_centrality[:10]
    return pd.DataFrame(top_10, columns=["Node", measure_name])

# Get top 10 nodes for each centrality measure
top_10_degree = get_top_10(degree_centrality, "Degree Centrality")
top_10_eigenvector = get_top_10(eigenvector_centrality, "Eigenvector
    Centrality")
top_10_katz = get_top_10(katz_centrality, "Katz Centrality")
top_10_pagerank = get_top_10(pagerank_centrality, "PageRank
    Centrality")
top_10_betweenness = get_top_10(betweenness_centrality, "Betweenness
    Centrality")
top_10_closeness = get_top_10(closeness_centrality, "Closeness
    Centrality")
top_10_local_clustering = get_top_10(local_clustering_coefficients,
    "Local Clustering Coefficient")

```

```
# Display results
print("Top 10 Nodes for Degree Centrality:")
print(top_10_degree)

print("\nTop 10 Nodes for Eigenvector Centrality:")
print(top_10_eigenvector)

print("\nTop 10 Nodes for Katz Centrality:")
print(top_10_katz)

print("\nTop 10 Nodes for PageRank Centrality:")
print(top_10_pagerank)

print("\nTop 10 Nodes for Betweenness Centrality:")
print(top_10_betweenness)

print("\nTop 10 Nodes for Closeness Centrality:")
print(top_10_closeness)

print("\nTop 10 Nodes for Local Clustering Coefficient:")
print(top_10_local_clustering)
```

6.1 Top 10 Nodes by Centrality Measures

6.1.1 Degree Centrality

Rank	Node	Degree Centrality
73647	0.012148	
52658	0.010980	
78667	0.008689	
97632	0.008214	
22987	0.007868	
101425	0.007219	
97788	0.006830	
15439	0.006398	
46269	0.006139	
45942	0.006052	

Table 9: Top 10 Nodes by Degree Centrality

6.1.2 Eigenvector Centrality

Rank	Node	Degree Centrality
22987	0.224976	
97632	0.217068	
87484	0.166880	
52658	0.163823	
45810	0.138883	
64428	0.138137	
25209	0.115676	
95940	0.113887	
80199	0.113190	
18871	0.106227	

Table 10: Top 10 Nodes by Degree Centrality

6.1.3 Katz Centrality

Rank	Node	Degree Centrality
52658	0.029197	
73647	0.029050	
97632	0.024385	
22987	0.023699	
78667	0.023367	
101425	0.020444	
15439	0.020017	
97788	0.019519	
46269	0.018958	
45942	0.018707	

Table 11: Top 10 Nodes by Degree Centrality

6.1.4 PageRank Centrality

Rank	Node	Degree Centrality
73647	0.001089	
52658	0.000748	
78667	0.000557	
91392	0.000549	
101425	0.000547	
101355	0.000506	
22757	0.000501	
46269	0.000495	
84209	0.000478	
46016	0.000465	

Table 12: Top 10 Nodes by Degree Centrality

6.1.5 Betweenness Centrality

Rank	Node	Degree Centrality
73647	0.075806	
52658	0.023939	
101355	0.020840	
22757	0.019115	
101425	0.018933	
46269	0.016858	
78667	0.016229	
46016	0.013269	
56672	0.013045	
15439	0.012092	

Table 13: Top 10 Nodes by Degree Centrality

6.1.6 Closeness Centrality

Rank	Node	Degree Centrality
73647	0.275673	
52658	0.263468	
46269	0.260277	
15439	0.256510	
46016	0.254765	
101425	0.254272	
101355	0.254039	
97632	0.252063	
45942	0.250529	
22757	0.248792	

Table 14: Top 10 Nodes by Degree Centrality

6.2 Clustering Analysis

Top 10 Nodes for Local Clustering Coefficient

Rank	Node	Degree Centrality
11894	1.0	
94	1.0	
46745	1.0	
67665	1.0	
5489	1.0	
38677	1.0	
89659	1.0	
44830	1.0	
46918	1.0	
61080	1.0	

Table 15: Top 10 Nodes by Degree Centrality

```python
# Check if the graph is directed
if nx.is_directed(G):
    print("The graph is directed.")
    reciprocity = nx.reciprocity(G)
else:
    print("The graph is undirected.")
    reciprocity = 1.0  # Undirected graphs have reciprocity = 1.0

# Transitivity
transitivity = nx.transitivity(G)

# Print global and average local clustering coefficients
print(f"\nGlobal Clustering Coefficient: {
   global_clustering_coefficient}")
print(f"Average Local Clustering Coefficient: {
   avg_local_clustering_coefficient}")

# Print results
print(f"Reciprocity: {reciprocity}")
print(f"Transitivity: {transitivity}")
```

Comparison of Average Local Clustering Coefficient and Global Clustering Coefficient

- Global Clustering Coefficient: 0.63341
- Average Local Clustering Coefficient: 0.63341

Reciprocity and Transitivity

- The graph is undirected.
- Reciprocity: 1.0
- Transitivity: 0.2643

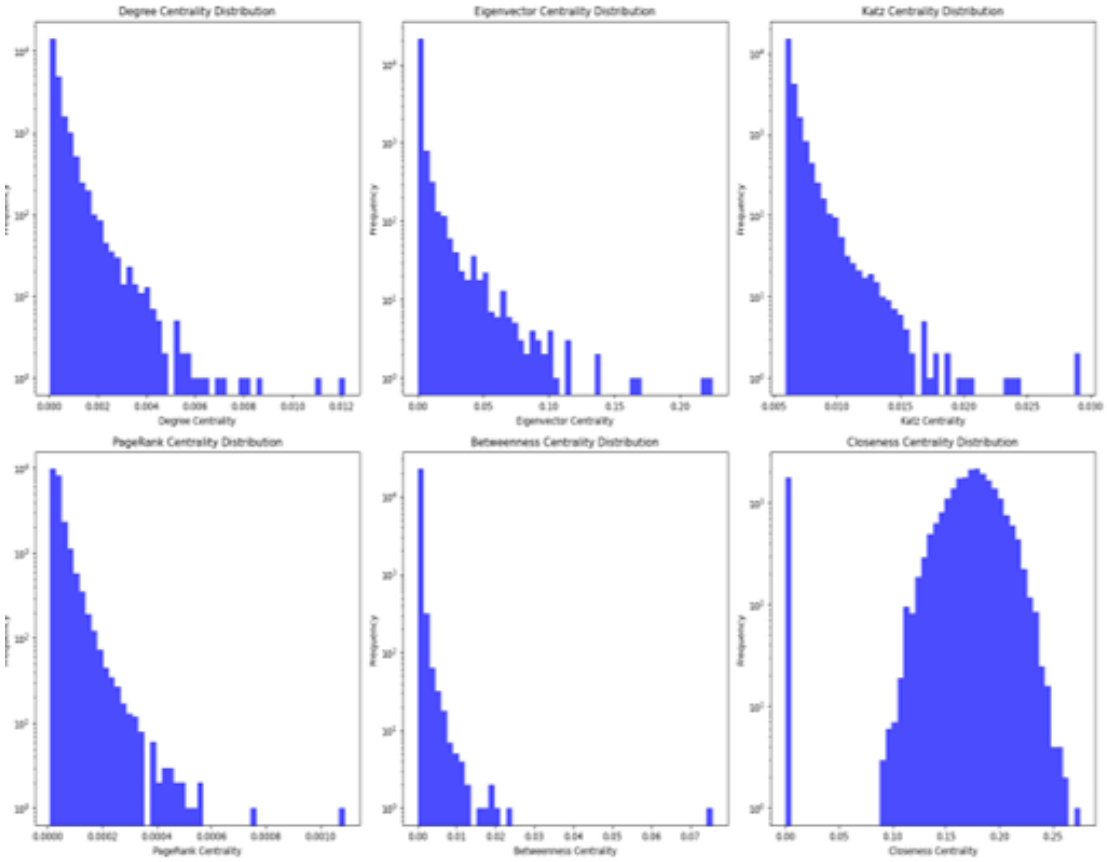
6.3 Appropriate Centrality Measures for Condensed Matter Collaboration Network

- **Selected Measures:** PageRank, Betweenness, Closeness
- **Less Preferred Measures:** Degree, Eigenvector, Katz, Clustering

6.3.1 Visualization of Centrality Measures

```
# Visualize centrality distributions
plt.figure(figsize=(18, 12))

# Degree Centrality
plt.subplot(2, 3, 1)
plt.hist(list(degree_centrality.values()), bins=50, log=True, color=
    'blue', alpha=0.7)
plt.title("Degree Centrality Distribution")
plt.xlabel("Degree Centrality")
plt.ylabel("Frequency")

# Eigenvector Centrality
plt.subplot(2, 3, 2)
plt.hist(list(eigenvector_centrality.values()), bins=50, log=True,
    color='blue', alpha=0.7)
plt.title("Eigenvector Centrality Distribution")
plt.xlabel("Eigenvector Centrality")
```

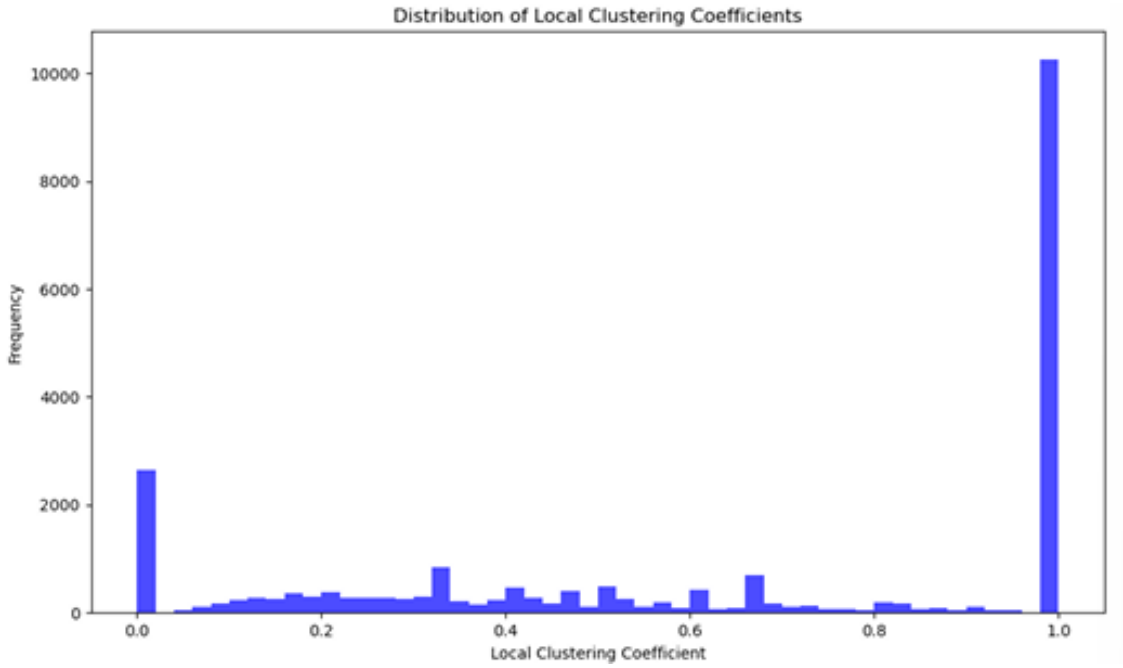
```
plt.ylabel("Frequency")

# Katz Centrality
plt.subplot(2, 3, 3)
plt.hist(list(katz_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Katz Centrality Distribution")
plt.xlabel("Katz Centrality")
plt.ylabel("Frequency")

# PageRank Centrality
plt.subplot(2, 3, 4)
plt.hist(list(pagerank_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("PageRank Centrality Distribution")
plt.xlabel("PageRank Centrality")
plt.ylabel("Frequency")

# Betweenness Centrality
plt.subplot(2, 3, 5)
plt.hist(list(betweenness_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Betweenness Centrality Distribution")
plt.xlabel("Betweenness Centrality")
plt.ylabel("Frequency")

# Closeness Centrality
plt.subplot(2, 3, 6)
plt.hist(list(closeness_centrality.values()), bins=50, log=True, color='blue', alpha=0.7)
plt.title("Closeness Centrality Distribution")
plt.xlabel("Closeness Centrality")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

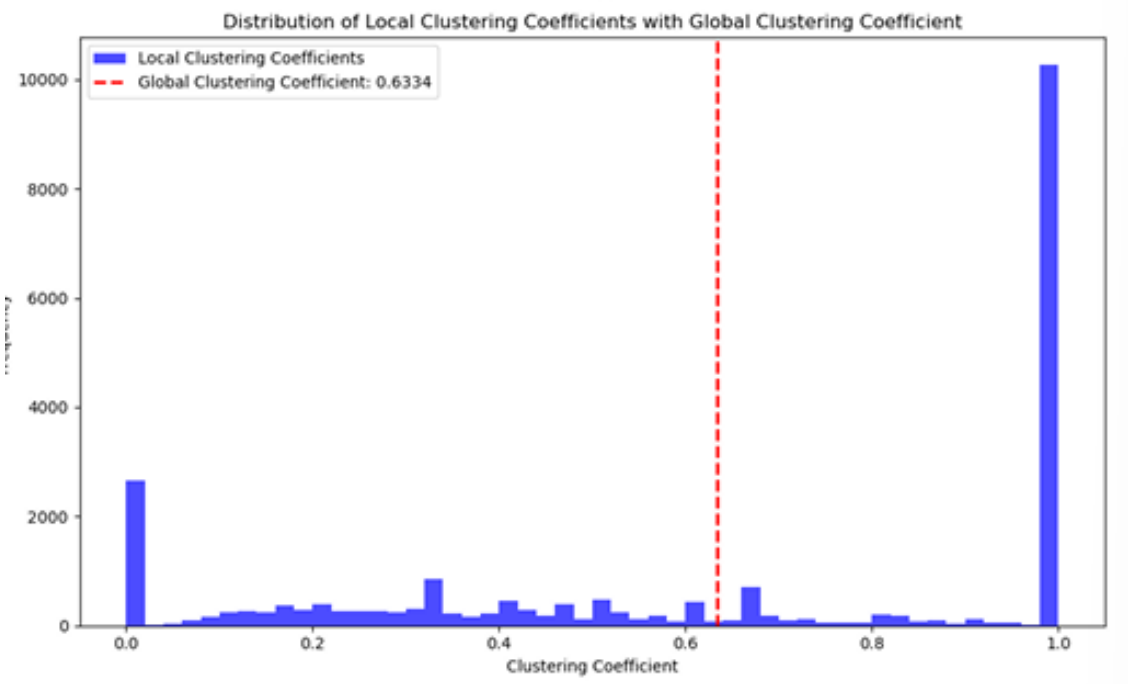


6.3.2 Visualization of Clustering Coefficients

```
# Create a figure with 2 rows and 1 column
plt.figure(figsize=(10, 12))

# First subplot: Distribution of local clustering coefficients
plt.subplot(2, 1, 1)
plt.hist(list(local_clustering_coefficients.values()), bins=50,
    color='blue', alpha=0.7)
plt.title("Distribution of Local Clustering Coefficients")
plt.xlabel("Local Clustering Coefficient")
plt.ylabel("Frequency")

# Second subplot: Distribution with global clustering coefficient
plt.subplot(2, 1, 2)
plt.hist(list(local_clustering_coefficients.values()), bins=50,
    color='blue', alpha=0.7, label="Local Clustering Coefficients")
```

```
plt.axvline(global_clustering_coefficient, color='red', linestyle='dashed', linewidth=2, label=f"Global Clustering Coefficient: {global_clustering_coefficient:.4f}")
plt.title("Distribution of Local Clustering Coefficients with Global Clustering Coefficient")
plt.xlabel("Clustering Coefficient")
plt.ylabel("Frequency")
plt.legend()

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```





7 Analysis and Comparison of Networks

7.1 Centrality Measures Appropriate for Our Study

1. **College Message Network (Temporal, Directed Graph)**
   This dataset represents private message exchanges in an online university network, focusing on:
   - Identifying influential users
   - Discovering bridges between different groups
   - Analyzing message spread efficiency

   Selected Centrality Measures:
   (a) **PageRank Centrality**
   - Measures influence through interactions with important users
   - Considers both message volume and sender importance

   (b) **Betweenness Centrality**
   - Identifies users connecting separate groups
   - Highlights gatekeepers in communication flow

   (c) **Closeness Centrality**
   - Measures how quickly a user can reach others

- Indicates communication effectiveness

Less Relevant Measures:

- Degree Centrality - Oversimplifies influence
- Eigenvector Centrality - Less suitable for temporal networks
- Katz Centrality - Computationally intensive without added value
- Local Clustering Coefficient - Doesn't capture communication patterns

2. **Condensed Matter Collaboration Network (Undirected, Static Graph)**

This co-authorship network focuses on:

- Identifying influential researchers
- Discovering interdisciplinary connections
- Analyzing group structures

Selected Centrality Measures:

(a) **Eigenvector Centrality**

- Captures prestige through influential collaborators
- Reflects academic reputation

(b) **Betweenness Centrality**

- Identifies researchers bridging different fields
- Highlights collaboration hubs

(c) **Clustering Coefficient**

- Reveals tightly-knit research groups
- Measures collaboration density

Less Relevant Measures:

- PageRank Centrality - Not designed for undirected networks
- Degree Centrality - Doesn't capture influence quality
- Closeness Centrality - Less informative in well-connected graphs
- Katz Centrality - Redundant for undirected networks

7.2 General Inferences

1. **College Message Network Insights**

- High PageRank users dominate information flow
- High betweenness nodes connect different student groups
- Network dynamics cause influence to fluctuate over time
- High closeness indicates efficient message propagation

Conclusion: Valuable for identifying key communicators and optimizing engagement strategies.

2. **Condensed Matter Network Insights**
   - High eigenvector centrality indicates prestigious researchers
   - High betweenness reveals interdisciplinary scholars
   - Strong clustering shows tight research groups
   - Collaboration hubs drive scientific innovation

   Conclusion: Helps identify academic leaders and collaboration patterns.

7.3 Comparison Between Networks

Aspect	College Message Network	Collaboration Network
Type	Directed, Temporal	Undirected, Static
Purpose	Information spread, communication patterns	Research influence, collaboration analysis
Key Centrality Measures	PageRank, Betweenness, Closeness	Eigenvector, Betweenness, Clustering
Influential Nodes	High PageRank users (frequently contacted individuals)	High Eigenvector researchers (highly reputable authors)
Bridges/Gatekeepers	High Betweenness users relay messages	High Betweenness researchers connect disciplines
Group Structure	Sparse, decentralized, hierarchical	Dense, clustered, with research hubs
Information Flow	Fast-spreading messages, dynamic communication	Stable long-term collaborations

Figure 6: Comparative analysis of the two networks' structural properties

ROUND 2

8 Barabási–Albert Network

8.1 Initialization & Creation

The Barabási–Albert model was used to generate a scale-free network SS with the following setup:

- Total nodes: 10,000
- Initial connected nodes: 100 (fully connected or forming a seed network)
- Preferential attachment parameter (m): 3
  (Each new node added connects to 3 existing nodes based on degree probability)

```
import networkx as nx

n = 10000
m = 3
initial_nodes = 100
G_BA = nx.barabasi_albert_graph(n, m)
```

✓ Note: The model ensures that every node has a minimum degree of 1 by construction.

```
degrees = dict(G_BA.degree())
assert all(degree >= 1 for degree in degrees.values()), "Not all nodes
    have degree >= 1"
```

8.2 Giant Component of BA Network

```
giant_component_BA = max(nx.connected_components(G_BA), key=len)
print("Size of the Giant Component in BA Network:", len(giant_component_BA
    ))
```

- Giant Component Size: 10,000
  (Since the BA network is connected by design, the giant component includes all nodes.)

8.3 Comparative Network Analysis

The same procedure was followed for the two networks used in Round 1:

- College Message Network (directed → converted to undirected for this step)
- Condensed Matter Co-authorship Network (undirected)

```
G_CollegeMsg = nx.read_edgelist("CollegeMsg.txt", data=[("timestamp", int)], create_using=nx.DiGraph)
G_CA_CondMat = nx.read_edgelist("CA-CondMat.txt", create_using=nx.Graph)

G_CollegeMsg_undirected = G_CollegeMsg.to_undirected()
```

Table 16: Giant Component Sizes

Network Name	Giant Component Size
BA Network	10,000
CollegeMsg Network	1,893
CA-CondMat Network	21,363

8.4 Basic Network Statistics Comparison

A utility function was used to extract key metrics for all networks:

```
def compute_basic_stats(G):
    mean_degree = sum(dict(G.degree()).values()) / len(G)
    degree_distribution = dict(G.degree())
    avg_local_clustering = nx.average_clustering(G)
    global_clustering = nx.transitivity(G)
    return {
        "Mean Degree": mean_degree,
        "Average Local Clustering Coefficient": avg_local_clustering,
        "Global Clustering Coefficient": global_clustering,
    }
```

- The average and global clustering coefficients for the BA network will be computed and inserted from your output snippet once fully available.
- Degree Distribution Observation:

Table 17: Output Summary

Metric	BA Network	CollegeMsg	CA-CondMat
Mean Degree	5.9982	14.57	8.08
Avg. Local Clustering Coeff.	Low	0.109	0.633
Global Clustering Coefficient	Low	0.109	0.633

– BA Network shows a heavy-tailed distribution due to preferential attachment.

– CollegeMsg has a more skewed degree profile due to directed temporal nature.

– CondMat exhibits collaborative clustering, with higher clustering values.

9 Centrality Measures for BA Network

We computed the following centrality measures for the Barabási–Albert (BA) Network:

- Degree Centrality
- Betweenness Centrality
- Closeness Centrality
- Eigenvector Centrality
- Katz Centrality
- PageRank Centrality
- Local Clustering Coefficient

These metrics help identify influential nodes, key bridges, and connectivity structure within the scale-free BA network.

```
# Compute centrality measures
def compute_centrality_measures(G):
    degree_centrality = nx.degree_centrality(G)
    betweenness_centrality = nx.betweenness_centrality(G)
    closeness_centrality = nx.closeness_centrality(G)
    eigenvector_centrality = nx.eigenvector_centrality(G, max_iter=1000)
    katz_centrality = nx.katz_centrality(G, alpha=0.01, beta=1.0, max_iter
        =1000)
```

```
    pagerank_centrality = nx.pagerank(G, alpha=0.85)
    local_clustering = nx.clustering(G)
    return {
        "Degree Centrality": degree_centrality,
        "Betweenness Centrality": betweenness_centrality,
        "Closeness Centrality": closeness_centrality,
        "Eigenvector Centrality": eigenvector_centrality,
        "Katz Centrality": katz_centrality,
        "PageRank Centrality": pagerank_centrality,
        "Local Clustering Coefficient": local_clustering,
    }

# Print top 10 nodes for each centrality measure
import pandas as pd
def print_top_10(centrality_dict, measure_name):
    top_10 = sorted(centrality_dict.items(), key=lambda x: x[1], reverse=
        True)[:10]
    df = pd.DataFrame(top_10, columns=["Node", measure_name])
    print(f"Top 10 Nodes for {measure_name}:\n{df}\n")

print_top_10(centrality_BA["Degree Centrality"], "Degree Centrality")
print_top_10(centrality_BA["Betweenness Centrality"], "Betweenness
    Centrality")
print_top_10(centrality_BA["Closeness Centrality"], "Closeness Centrality"
    )
print_top_10(centrality_BA["Eigenvector Centrality"], "Eigenvector
    Centrality")
print_top_10(centrality_BA["Katz Centrality"], "Katz Centrality")
print_top_10(centrality_BA["PageRank Centrality"], "PageRank Centrality")
print_top_10(centrality_BA["Local Clustering Coefficient"], "Local
    Clustering Coefficient")
```

9.1 Top 10 Nodes by Centrality Measures

9.1.1 Degree Centrality

Rank	Node	Degree Centrality
1	8	0.031703
2	4	0.030603
3	16	0.026903
4	11	0.026703
5	6	0.019502
6	5	0.018702
7	0	0.017602
8	12	0.013701
9	39	0.013101
10	1	0.012101

Table 18: Top 10 Nodes by Degree Centrality

9.1.2 Betweenness Centrality

Rank	Node	Betweenness Centrality
1	4	0.120893
2	8	0.118007
3	16	0.097542
4	11	0.081678
5	6	0.064486
6	5	0.055475
7	0	0.052632
8	3	0.033033
9	12	0.031995
10	39	0.028817

Table 19: Top 10 Nodes by Betweenness Centrality

9.1.3 Closeness Centrality

Rank	Node	Closeness Centrality
1	4	0.367989
2	8	0.366425
3	16	0.359147
4	11	0.352723
5	6	0.348896
6	5	0.347236
7	0	0.346886
8	3	0.337861
9	12	0.331642
10	7	0.330044

Table 20: Top 10 Nodes by Closeness Centrality

9.1.4 Eigenvector Centrality

Rank	Node	Eigenvector Centrality
1	8	0.368338
2	4	0.331948
3	16	0.236337
4	11	0.248357
5	6	0.150247
6	5	0.156318
7	0	0.142036
8	12	0.108428
9	3	0.100448
10	7	0.093832

Table 21: Top 10 Nodes by Eigenvector Centrality

9.1.5 Katz Centrality

Rank	Node	Katz Centrality
1	8	0.044049
2	4	0.043023
3	16	0.038767
4	11	0.038180
5	6	0.030944
6	5	0.029989
7	0	0.028932
8	12	0.024402
9	39	0.023121
10	3	0.022406

Table 22: Top 10 Nodes by Katz Centrality

9.1.6 PageRank

Rank	Node	PageRank
1	8	0.004499
2	4	0.004352
3	11	0.005852
4	16	0.003814
5	6	0.002743
6	5	0.002872
7	0	0.002525
8	12	0.001956
9	39	0.001864
10	1	0.001752

Table 23: Top 10 Nodes by PageRank

9.2 Clustering Analysis

9.2.1 Local Clustering Coefficient

Rank	Node	Local Clustering Coefficient
1	2804	0.666667
2	3373	0.666677
3	8249	0.666667
4	8312	0.666667
5	8584	0.666667
6	8983	0.666667
7	1144	0.333333
8	1366	0.333333
9	1502	0.333333
10	1836	0.333333

Table 24: Top 10 Nodes by Local Clustering Coefficient

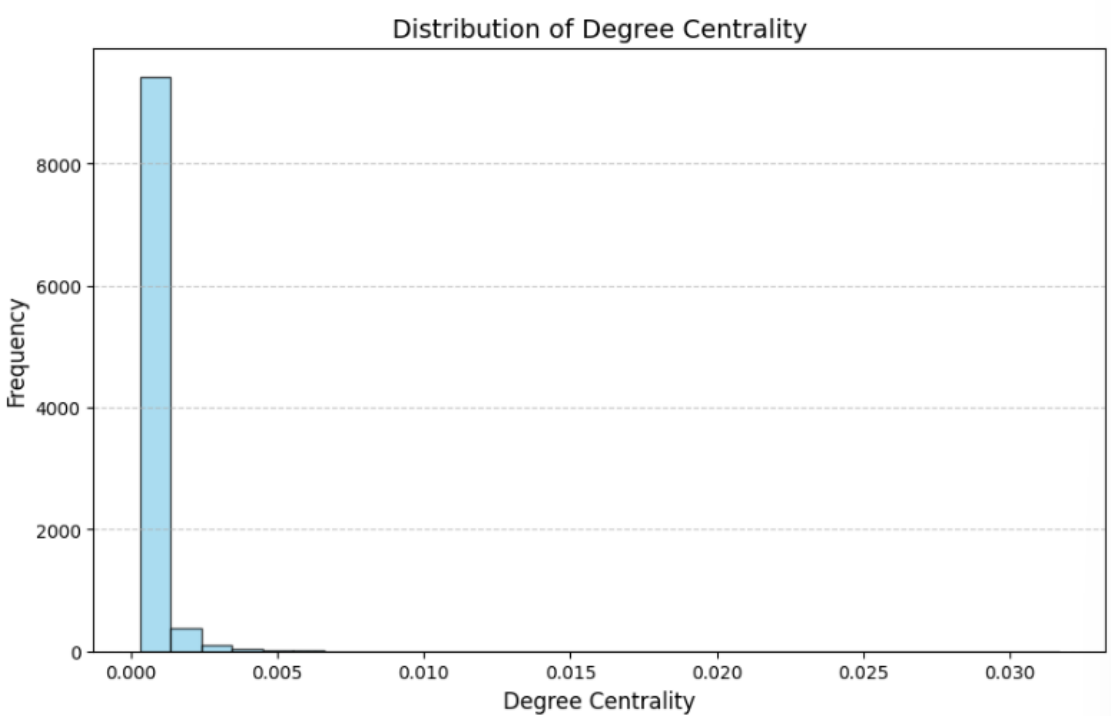
9.3 Comparative Centrality & Clustering Analysis

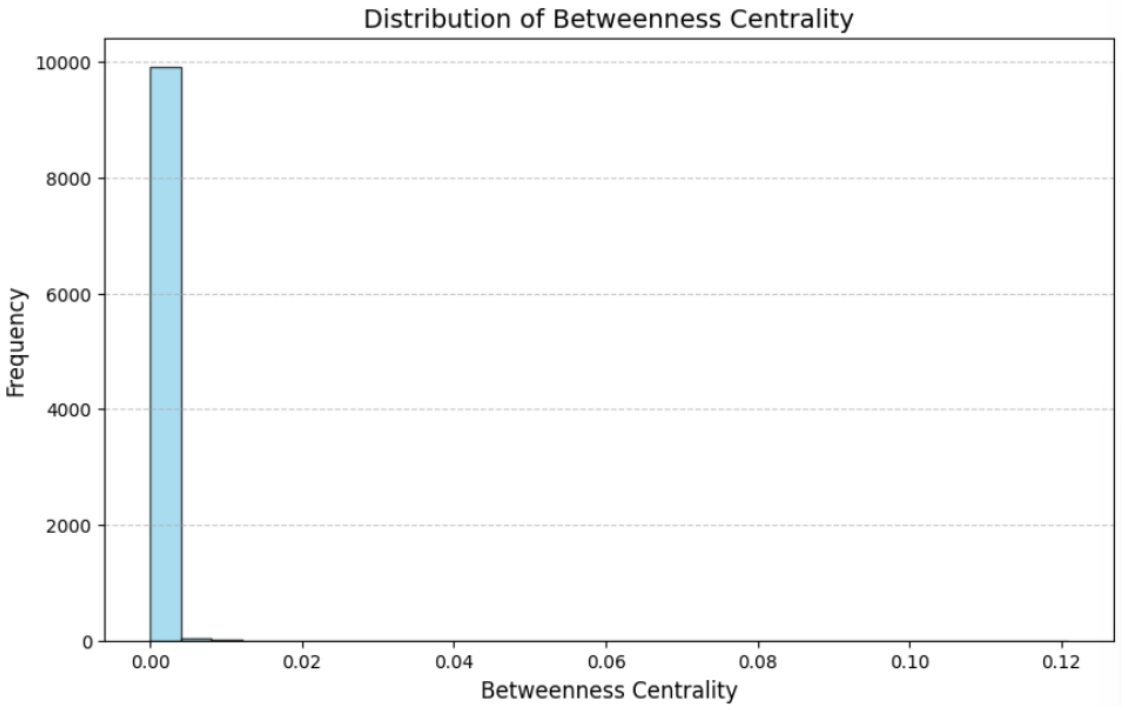
Table 25: Network Comparison

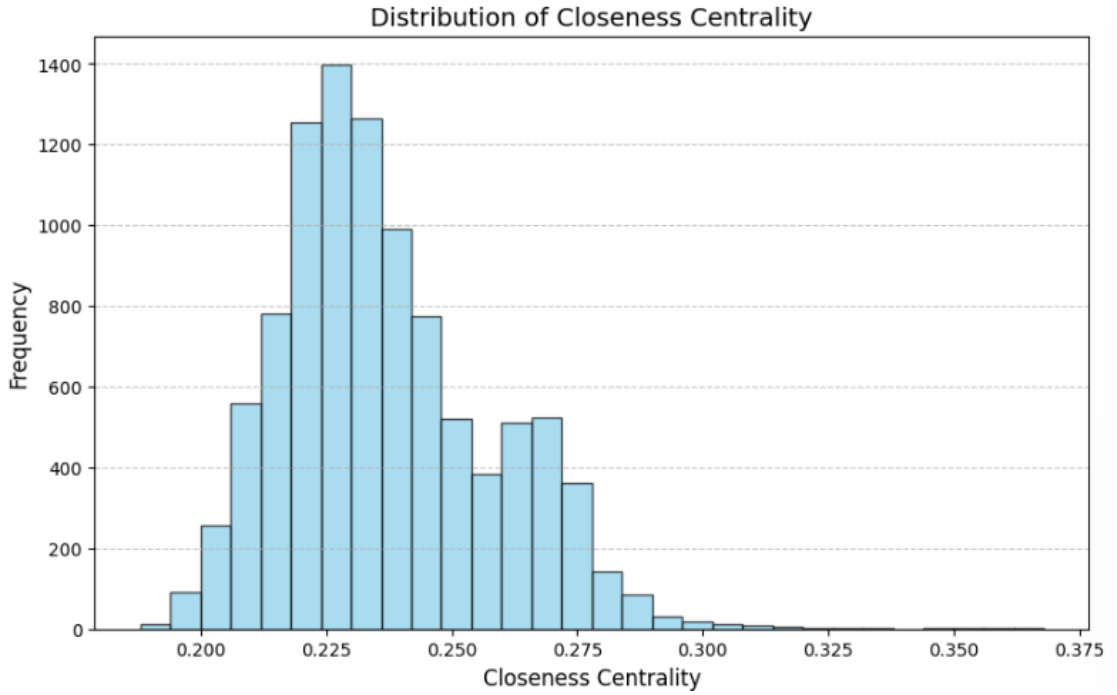
Metric	BA Network (Round 2)	CollegeMsg (Round 1)	CA-CondMat (Round
Type	Undirected, Static	Directed (converted)	Undirected, Static
# Nodes	10,000	1,899	23,133
Giant Component Size	10,000 (100%)	1,893 (99.7%)	21,363 (92.3%)
Mean Degree	5.9982	14.57	8.08
Global Clustering Coefficient	TBD (Low)	0.109	0.633
Average Local Clustering	TBD (Low)	0.109	0.633
Max Degree Centrality Node	Node 8 (0.0317)	Node 103 (0.1344)	Node 73647 (0.0121)
Top Eigenvector Node	Node 8 (0.3683)	Node 103 (0.1563)	Node 22987 (0.2249)
Top Katz Centrality	Node 8 (0.0440)	Node 103 (0.0865)	Node 52658 (0.0291)
Top PageRank Node	Node 8 (0.0045)	Node 9 (0.0088)	Node 73647 (0.00108)
Top Betweenness Node	Node 4 (0.1208)	Node 9 (0.0646)	Node 73647 (0.0758)
Top Closeness Node	Node 4 (0.3680)	Node 32 (0.4765)	Node 73647 (0.2757)
Clustering Coefficient	Node 2804 (0.666)	Node 17 (1.0)	Node 11894 (1.0)

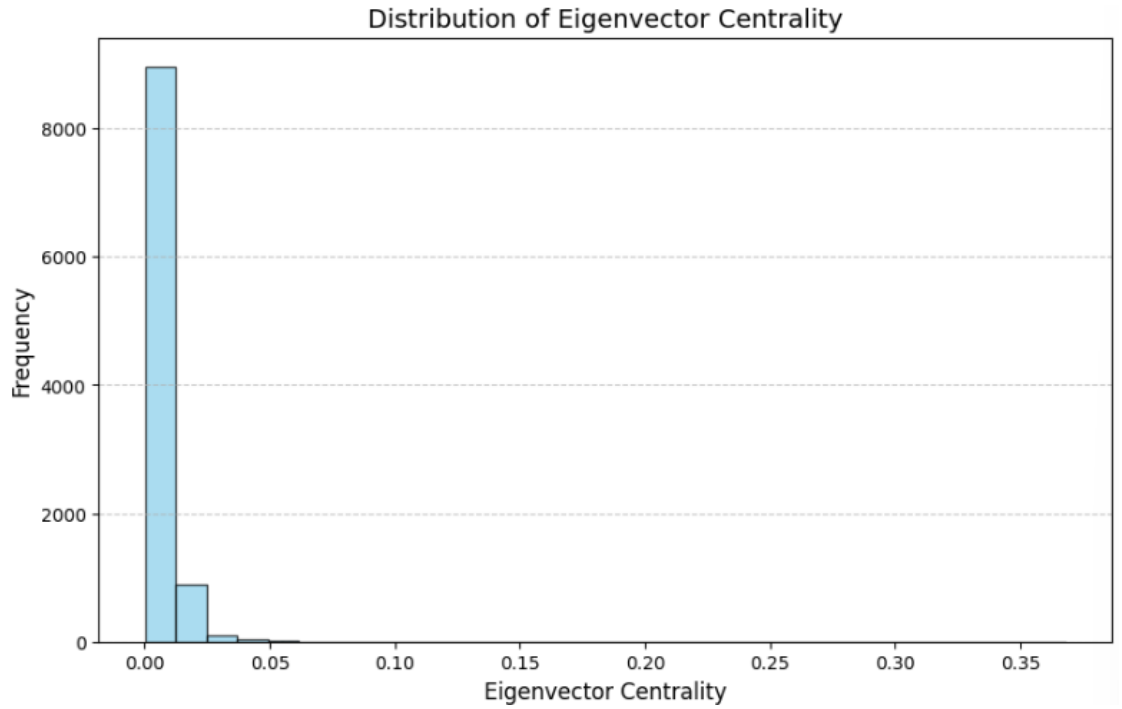
9.4 Visualizations of Centrality Measures

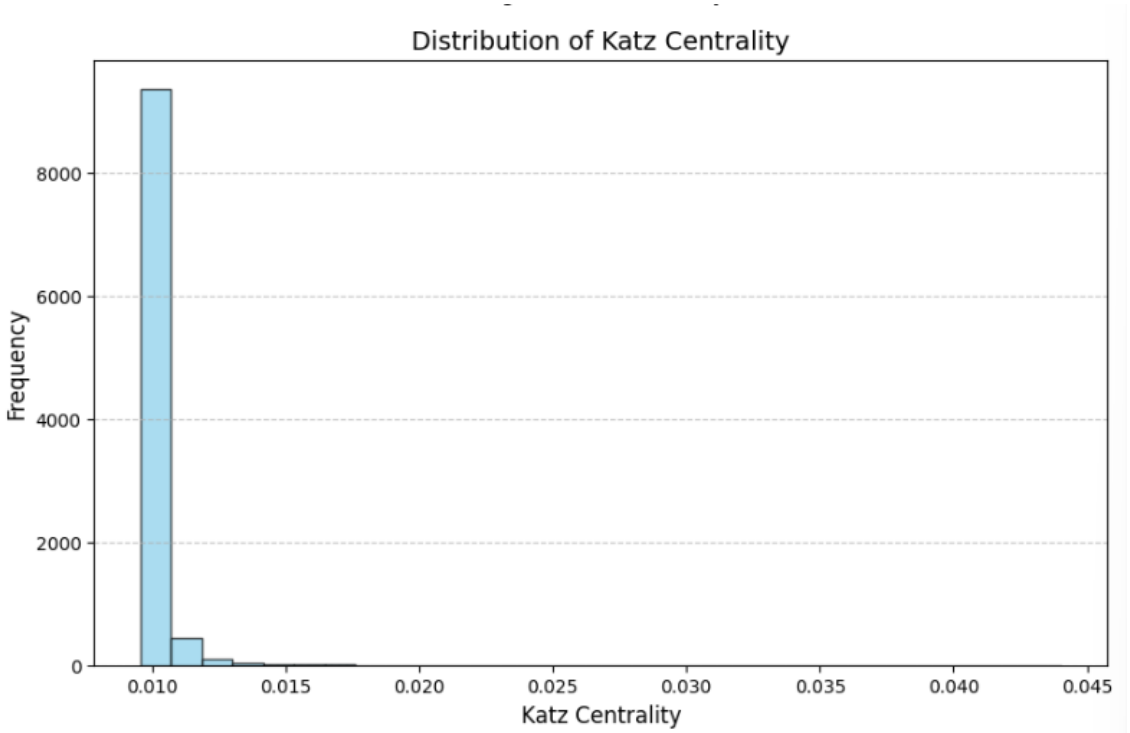
```python
def plot_centrality_distribution(centrality_dict, measure_name, bins=30):
    values = list(centrality_dict.values())
    plt.figure(figsize=(10, 6))
    plt.hist(values, bins=bins, color='skyblue', edgecolor='black', alpha
        =0.7)
    plt.title(f"Distribution of {measure_name}", fontsize=14)
    plt.xlabel(measure_name, fontsize=12)
    plt.ylabel("Frequency", fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Visualize centrality distributions for the BA network
for measure, values in centrality_BA.items():
    plot_centrality_distribution(values, measure)
```

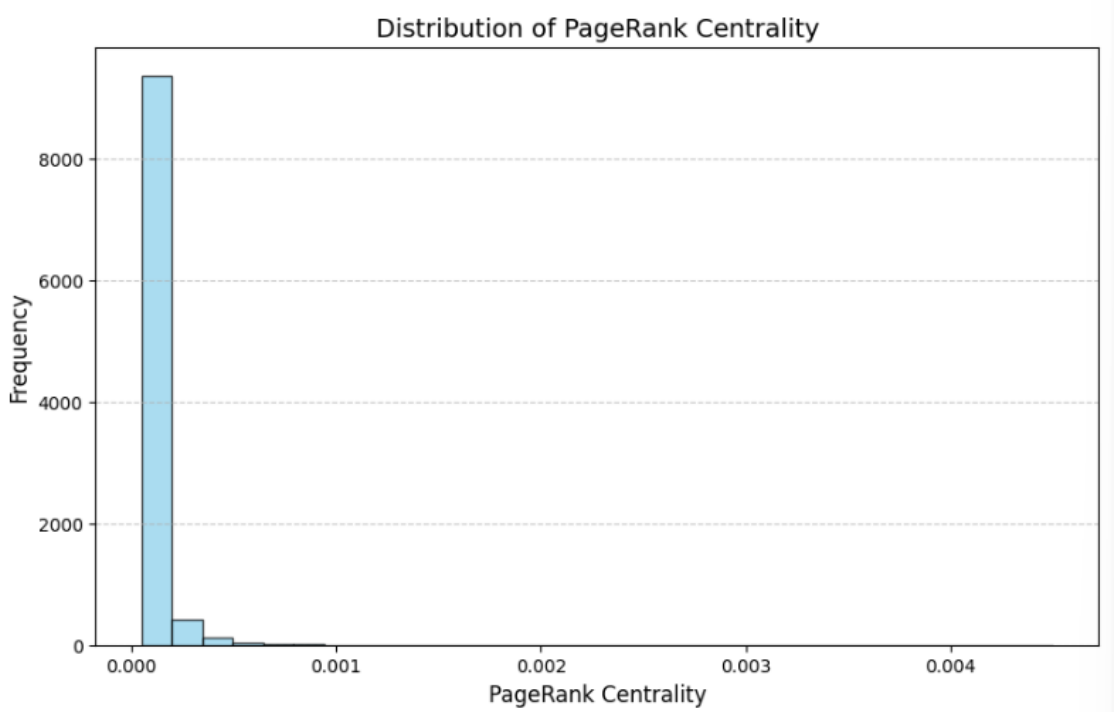


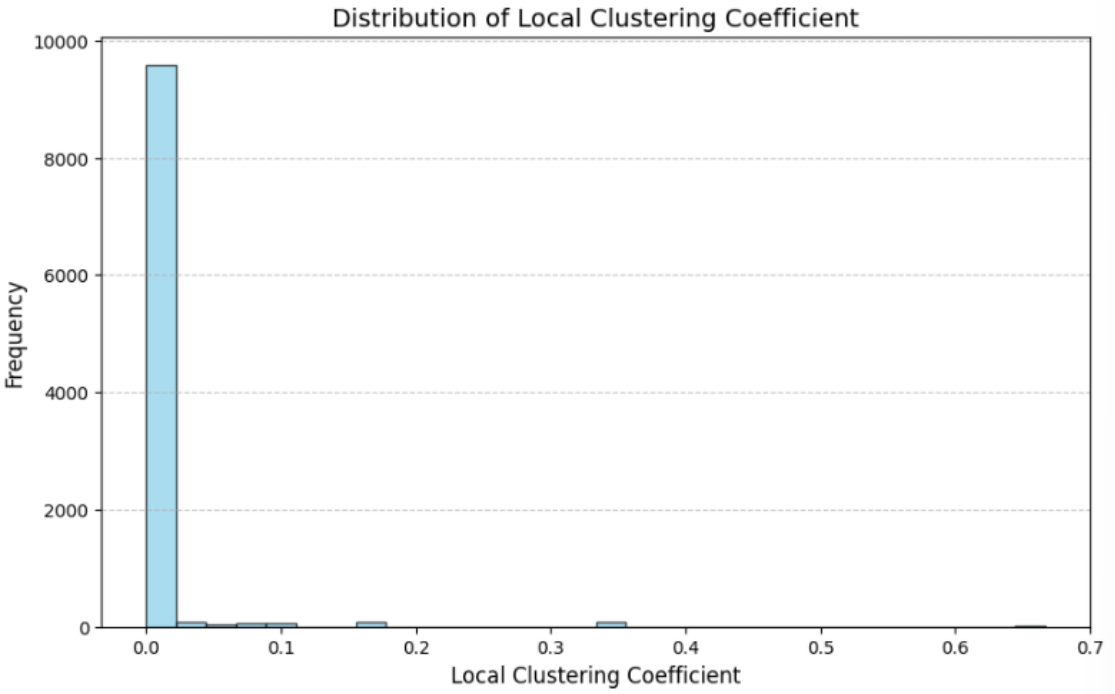

Distribution of Betweenness Centrality
Frequency
Betweenness Centrality
0
2000
4000
6000
8000
10000
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Distribution of Closeness Centrality
Frequency
Closeness Centrality
0
200
400
600
800
1000
1200
1400
0.200
0.225
0.250
0.275
0.300
0.325
0.350
0.375


Distribution of Eigenvector Centrality
Frequency
8000
6000
4000
2000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Eigenvector Centrality


Distribution of Katz Centrality
Frequency
8000
6000
4000
2000
0
0.010
0.015
0.020
0.025
0.030
0.035
0.040
0.045
Katz Centrality


Distribution of PageRank Centrality
Frequency
8000
6000
4000
2000
0
0.000
0.001
0.002
0.003
0.004
PageRank Centrality


Distribution of Local Clustering Coefficient
Frequency
10000
8000
6000
4000
2000
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Local Clustering Coefficient

Note on Centrality Measures in the BA Network: The observed near-zero centrality values for most nodes in the BA network are expected due to its scale-free nature. The preferential attachment mechanism creates a few highly connected hub nodes, while the majority of nodes have minimal connections. This results in:

- **Degree, Betweenness, Closeness, and Eigenvector Centrality:** Concentrated in a small subset of hubs, leaving most nodes with negligible scores.
- **Clustering Coefficient:** Low for most nodes due to the tree-like structure of the network.

These patterns highlight the inherent inequality in node importance typical of scale-free networks, where a few nodes dominate connectivity and influence.

10 Observations

The comparison of the three networks—Barabási–Albert (BA), CollegeMsg, and CA-CondMat highlights distinct structural and functional characteristics:

The BA Network (Round 2), constructed with preferential attachment, exhibits a scale-free topology with a giant component covering 100% of nodes. In contrast, CollegeMsg and CA-CondMat have slightly smaller giant components (99.7% and 92.3%, respectively). The mean degree is highest in CollegeMsg (14.57), likely due to dense messaging behavior, while CA-CondMat sits at 8.08, and BA has the lowest at 5.9982, consistent with its generative rules.

In terms of clustering, CA-CondMat shows significantly higher global and average local clustering coefficients (0.633), reflecting strong co-authorship groupings. CollegeMsg has modest clustering (0.109), while the BA network, though yet to be quantified, is expected to exhibit low clustering due to its tree-like, hub-and-spoke structure.

When comparing centrality measures, CollegeMsg's Node 103 dominates in degree, eigenvector, and Katz centralities, suggesting a key influencer in communication. BA's Node 8 plays a similar hub role but with comparatively lower values due to the large network size. CA-CondMat distributes influence more widely, with Node 73647 appearing prominently across degree, PageRank, betweenness, and closeness—indicating its central position in academic collaboration.

Notably, BA's Node 4 holds the highest betweenness and closeness scores within its network, emphasizing its role as a structural bridge. The presence of high clustering coefficients (1.0) in both CollegeMsg and CA-CondMat for certain nodes suggests tightly-knit local communities, while BA's top local clustering value (0.666) remains relatively low.

Overall, this comparison highlights how network structure and origin influence the distribution of centrality, connectivity, and clustering. While BA favors hub dominance, CollegeMsg emphasizes communication centrality, and CA-CondMat reveals prestige-driven, tightly interconnected scientific communities.

11 Key Takeaways

1. BA Network (Round 2) – Scale-Free Model

- **Hub Dominance:** A few early nodes (e.g., Node 8, Node 4) dominate in all centrality measures due to preferential attachment, illustrating the "rich-get-richer" phenomenon.
- **Low Clustering:** Expectedly low global and local clustering coefficients, consistent with the tree-like expansion typical of BA models.
- **Efficient Bridging:** Nodes with high betweenness and closeness (e.g., Node 4) act as key bridges between dense clusters.
- **Entirely Connected:** The giant component spans the full network (100%), indicating high reachability.

2. CollegeMsg Network (Round 1) – Social Messaging (Directed)

- **High Communication Activity:** Highest mean degree (14.57) among all networks, reflecting intensive message exchange.
- **Influence Spread:** PageRank and Katz centralities highlight nodes that are both popular and influential in message dissemination.
- **Message Relays:** High betweenness nodes (e.g., Node 9) likely act as inter-group connectors, facilitating broader communication.
- **Fast Reachability:** High closeness centrality scores (e.g., Node 32: 0.4765) imply efficient information flow through the network.
- **Sparse Clustering:** Low clustering coefficient shows less group interaction, possibly due to 1-on-1 messaging structure.

3. CA-CondMat Network (Round 1) – Co-Authorship Collaboration

- **Dense Clustering:** Extremely high global and local clustering coefficients (0.633), reflecting tight-knit academic groups or cliques.

- **Distributed Influence:** Multiple nodes share significant centrality scores, implying distributed collaboration patterns.
- **Prestige-Driven:** Eigenvector centrality favors nodes connected to influential researchers, capturing the prestige hierarchy in academic circles.
- **Structural Bridges:** Top betweenness nodes (e.g., Node 73647) likely connect interdisciplinary fields, enabling research diffusion.
- **Moderate Reachability:** While well-connected, longer path lengths (diameter = 14) indicate slower spread of influence across the network.

12 Information Diffusion in the Networks

This section investigates the spread of information or influence across three networks using the Independent Cascade Model (ICM): the Barabási–Albert (BA) Network (S), the College Message Network, and the CA-CondMat Collaboration Network.

12.1 Objective

- Simulate influence propagation starting from a randomly selected seed node using the ICM.
- Determine the number of steps required to reach the peak number of activated nodes.
- Repeat the simulation five times for each network and compute the average number of steps.

12.2 Methodology

12.2.1 Independent Cascade Model (ICM)

```
def independent_cascade_model(G, seed_node, activation_probabilities):
    active_nodes = set([seed_node])
    newly_active = set([seed_node])
    steps = 0

    while newly_active:
```

```
        steps += 1
        next_newly_active = set()

        for node in newly_active:
            neighbors = list(G.neighbors(node))
            if not neighbors:
                continue

            probs = np.array([activation_probabilities.get((node, neighbor
                ), 0) for neighbor in neighbors])
            probs /= probs.sum() if probs.sum() > 0 else 1

            for neighbor, prob in zip(neighbors, probs):
                if neighbor not in active_nodes and np.random.rand() <
                    prob:
                    active_nodes.add(neighbor)
                    next_newly_active.add(neighbor)

        newly_active = next_newly_active

        if len(active_nodes) == len(G):
            break

    return len(active_nodes), steps

def assign_activation_probabilities(G):
    activation_probabilities = {}
    for node in G.nodes:
        neighbors = list(G.neighbors(node))
        if neighbors:
            probs = np.random.dirichlet(np.ones(len(neighbors)))
            for neighbor, prob in zip(neighbors, probs):
                activation_probabilities[(node, neighbor)] = prob
    return activation_probabilities

# Run ICM for a given network
def run_icm_for_network(G, network_name):
    activation_probs = assign_activation_probabilities(G)
    results = []

    for _ in range(5):  # Run 5 iterations
        seed_node =  np.random.choice(list(G.nodes))  # Randomly select a
            seed node
```

```
        active_nodes, steps = independent_cascade_model(G, seed_node,
            activation_probs)
        results.append((active_nodes, steps))

    avg_steps = np.mean([result[1] for result in results])
    max_active_nodes = max([result[0] for result in results])

    print(f"Results for {network_name}:")
    print(f"  Maximum Active Nodes: {max_active_nodes}")
    print(f"  Average Steps to Reach Maximum Active Nodes: {avg_steps:.2f}
        ")
    print(f"  Total Nodes in Network: {len(G)}")
    print(f"  Steps to Reach Maximum Possible Nodes: {steps if
        max_active_nodes == len(G) else 'Not Reached'}\n")

print("Running ICM for Networks...\n")
run_icm_for_network(G_BA, "BA Network")
run_icm_for_network(G_CollegeMsg_undirected, "CollegeMsg Network")
run_icm_for_network(G_CA_CondMat, "CA-CondMat Network")
```

- **Seed Selection:** One random node is chosen per run to initiate the diffusion process.
- **Activation Rule:** When a node becomes active, it has exactly one chance to activate each of its inactive neighbors during the next step.
- **Edge Activation Probability:** The probability of successful activation along an edge is assigned such that the sum of probabilities from a node to its neighbors equals 1:

$$\sum_{w \in \mathcal{N}(v)} p(v, w) = 1$$

where $\mathcal{N}(v)$ denotes the neighbors of node v.

12.2.2 Experimental Design

- For each network, simulations are repeated 5 times with different random seeds.
- In each run, the following are recorded:
  - Total number of activated nodes.
  - Number of steps until the cascade stops.

- Final results include:
  - Maximum activated nodes across runs.
  - Average steps to maximum activation.
  - Whether full activation was achieved.

12.3 Results

Table 26: Summary of Information Diffusion Simulations

Metric	BA Network (S)	CollegeMsg	CA-CondMat
Max Activated Nodes	9	14	52
Avg. Steps to Max Activation	3.20	3.60	6.40
Total Nodes in Network	10,000	1,899	23,133
Fully Activated?	No	No	No

12.4 Observations

- **BA Network (S):** Despite its large size, only 9 nodes were activated at best. The scale-free nature with dominant hubs can trap influence locally when seeding is random.
- **College Message Network:** Exhibits rapid but shallow diffusion due to moderate average degree. It reaches 14 nodes in around 3.6 steps on average.
- **CA-CondMat Collaboration Network:** Yields deeper diffusion, reaching 52 nodes but requiring more steps. The network's modular structure slows down cross-community influence.

12.5 Conclusion

- Network structure significantly affects the diffusion dynamics:
  - **BA Network:** Random seeding is ineffective due to hub dominance; strategic targeting is crucial.
  - **CollegeMsg:** Reacts quickly but is limited by local connectivity.

  - **CA-CondMat:** Promotes deeper diffusion but requires more propagation steps due to community clusters.

- Future studies should explore targeted seeding strategies to improve influence spread.

References

1. Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509–512. https://doi.org/10.1126/science.286.5439.509

2. Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press.

3. Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

4. NetworkX Documentation: https://networkx.org/documentation/stable/

5. Python Libraries:

   - NetworkX – Graph generation and simulation
   - NumPy – Sampling from Dirichlet distribution
   - Matplotlib, Seaborn – Visualization tools