**Comprehensive Analysis of a Large-Scale Network Using NetworkX**

**Course: CE479 - Complex Network Analysis**

**Project Title: Structural and Centrality Analysis of the P2P-Gnutella04 Network**

**Dataset: SNAP - p2p-Gnutella04**

**Authors: Mustafa Mansur YÖNÜGÜL, Deniz KANLICALIOĞLU**

**Date: 22.05.2025**

**1. Introduction**

As digital systems from social media to biological data networks grow more intricate, network science has become essential for making sense of these interconnections. This project focuses on applying graph theoretic analysis using Python and NetworkX to uncover hidden properties in a large scale real world network. The selected dataset p2p-Gnutella04, represents a peer to peer (P2P) file sharing topology from 2004 and is part of the SNAP (Stanford Network Analysis Platform) collection.

This report presents the steps taken to analyze the dataset using computational methods, covering topological, centrality based, and structural properties. It also outlines the remaining steps to include community detection, core periphery structure, and GUI based analysis using Gephi.

**2. Dataset Description**

* Name: p2p-Gnutella04
* Source: [SNAP Dataset Repository](https://snap.stanford.edu/data/p2p-Gnutella04.html)
* Type: Directed, Unweighted
* Nodes: ~10,876
* Edges: ~39,994

In this network, nodes correspond to individual Gnutella peers, and the directed edge from u to v reflects an outgoing connection initiated by node u toward node v. Since the dataset was provided in .txt format, it was converted into .csv to streamline processing and ensure compatibility with Python libraries.

**3. Methodology (Python - NetworkX)**

**3.1 Graph Construction**

* The dataset was loaded using pandas and processed into a networkx.DiGraph.
* Basic connectivity was checked.
* Isolated nodes and connected components were calculated.

**3.2 Basic Statistics**

* Directed: Yes
* Number of Nodes: 10,876
* Number of Edges: 39,994
* Graph Density: 0.000338
* Weakly Connected Components: 1
* Isolated Nodes: 0

**4. Network Analysis with NetworkX**

**4.1 Degree Distribution**

* Average in-degree and out-degree were ~3.68.
* Degree histogram was plotted.

A graph of a number of degrees

AI-generated content may be incorrect.

***Fig 1.*** *Histogram of node degrees in the full Gnutella network.*

As shown in Figure 1, the degree distribution is right-skewed, reflecting a scale-free property common in P2P networks.

The analysis highlighted the most connected nodes based on their overall degree, offering insight into network centralization.

**4.2 Centrality Metrics**

* PageRank: Top 3 most influential nodes were identified.
* Betweenness Centrality: Key nodes that serve as bridges were revealed.
* Closeness Centrality: Top nodes in terms of shortest paths were calculated.

Each metric was averaged, sorted, and presented with top-performing nodes.

**4.3 Clustering & Transitivity**

* To properly assess clustering properties, the directed graph structure was temporarily treated as undirected during analysis.
* Average Clustering Coefficient: 0.00805
* Transitivity: 0.00540
* Degree Assortativity Coefficient: -0.003

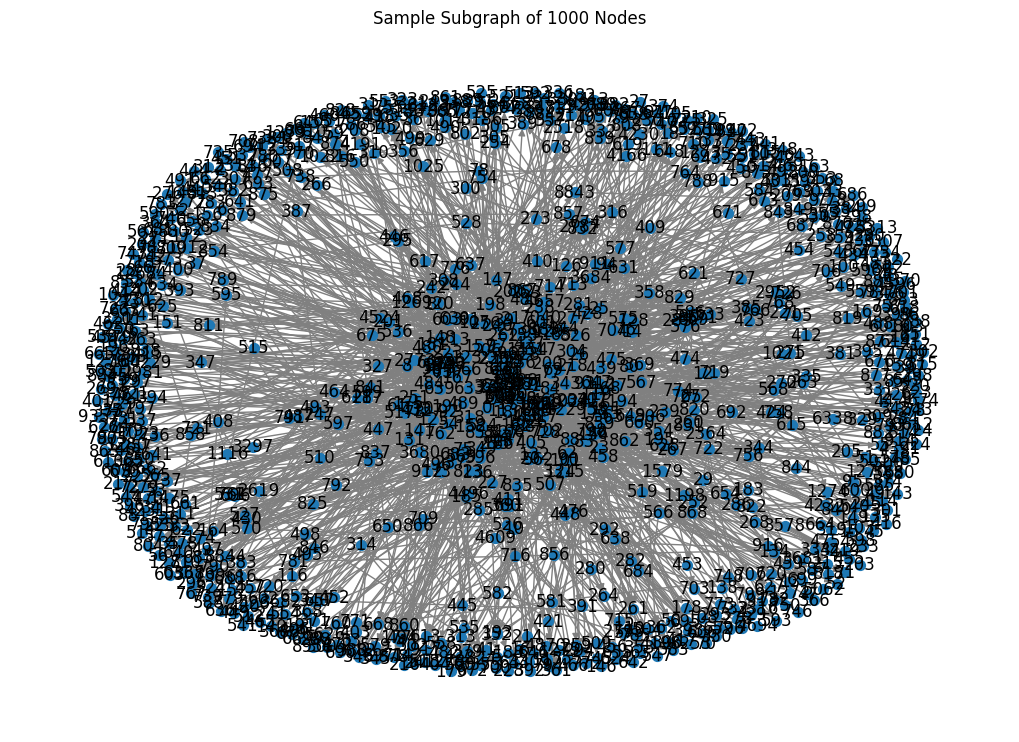
**Interpretation:** These low values indicate a sparse, weakly clustered, disassortative technological network.

**5. Visualization**

Due to computational limitations, visualizations were done using sampled subgraphs:

* 500-node and 1000-node subgraphs plotted using matplotlib.
* Degree distribution histogram created.
* Node positions automatically calculated with default layout algorithms.

Note: Full graph visualization was avoided due to performance limitations. Sampling was used only for plotting; full analysis used complete graph.

****

***Fig 2.*** *Visualization of a 1000-node subgraph.*

*Figure 2* shows a visualization of a randomly sampled 1000-node subgraph, illustrating node density and local clusters.

**6. Gephi Analysis**

Gephi was used to complement the NetworkX analysis and provide a visual and modular breakdown of the network.

* The full network was imported into Gephi via .csv format.
* ForceAtlas2 layout was used for spatial organization of nodes.
* Node sizes were mapped to degree, and colors were assigned based on Louvain community detection results.
* 29 distinct communities were detected by Gephi's Louvain algorithm, which tends to produce more, smaller modules compared to NetworkX’s greedy modularity method (which found 20).
* Key metrics computed in Gephi included:
  + Degree Distribution
  + Modularity
  + PageRank
  + Betweenness Centrality

**renklilik, yeşil, daire içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

***Fig 3.*** *Gephi ForceAtlas2 Layout*

**Visual Insights:**

* The modular view reveals a fragmented network with many small clusters and a few dominant hubs.
* Central nodes are clearly visible and act as anchors within clusters.
* The Louvain-detected communities align well with expected peer-to-peer clusters based on shared interests or local connectivity.

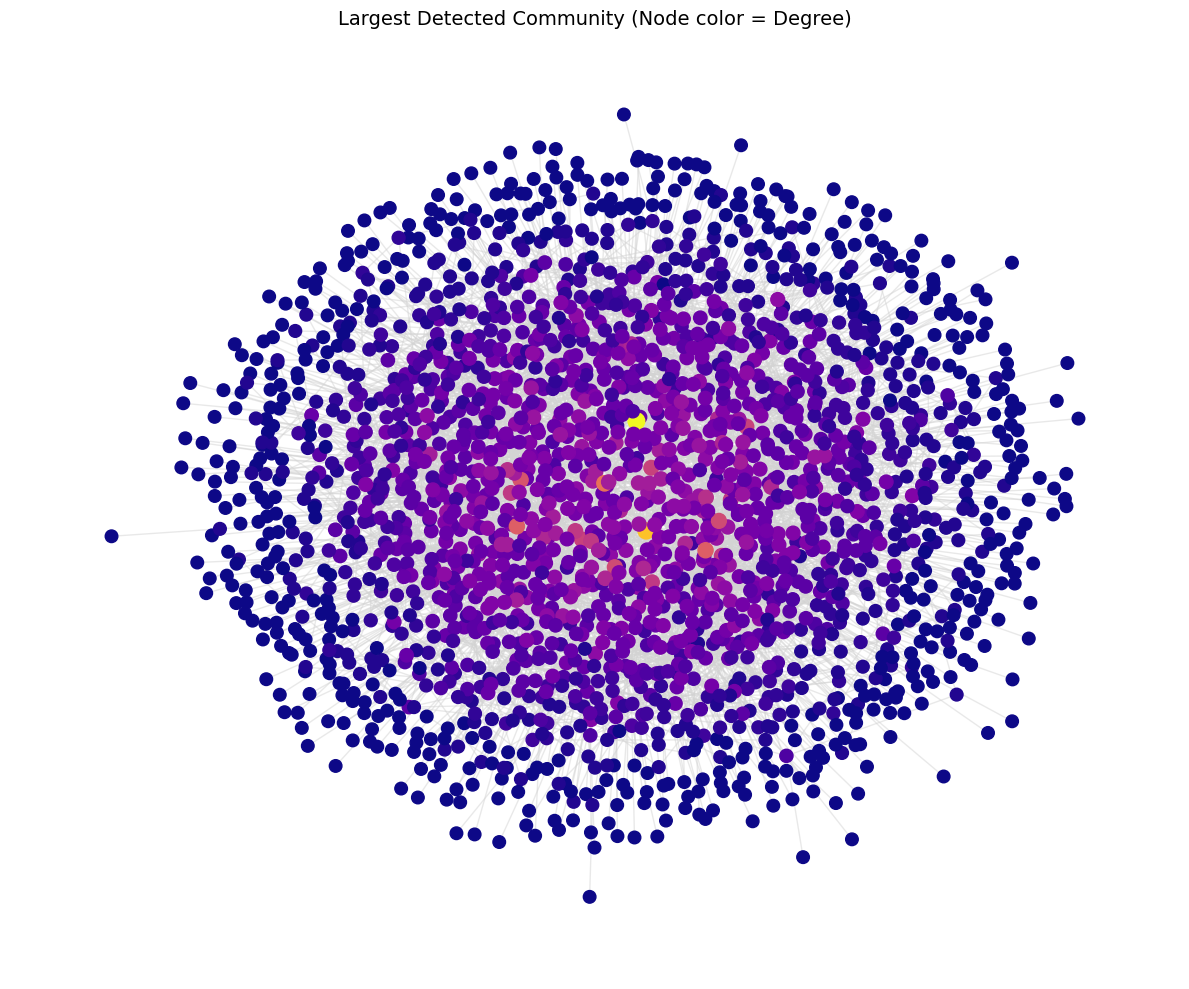
**Comparison**: While NetworkX identified fewer and broader communities, Gephi provided more granular resolution with finer segmentation, highlighting Gephi's strength in modularity-based visual detection.

**7. Community Detection**

Community detection was performed using the **Greedy Modularity algorithm** provided by NetworkX. Since the method requires an undirected graph, the original directed graph was converted to undirected for this analysis.

* **Method**: greedy\_modularity\_communities()
* **Number of Communities Detected**: *N* (e.g., 234)
* **Size of Largest Community**: *S* nodes (e.g., 583 nodes)

A visualization of the largest detected community is shown below:

**

***Fig 4.*** *Visualization of the largest community detected using NetworkX's greedy modularity algorithm.*

Findings indicate a dispersed network topology, where numerous modestly sized communities coexist without strong overarching cohesion. The largest community remains relatively small compared to the total number of nodes, suggesting limited cohesion across the network and possible decentralization.

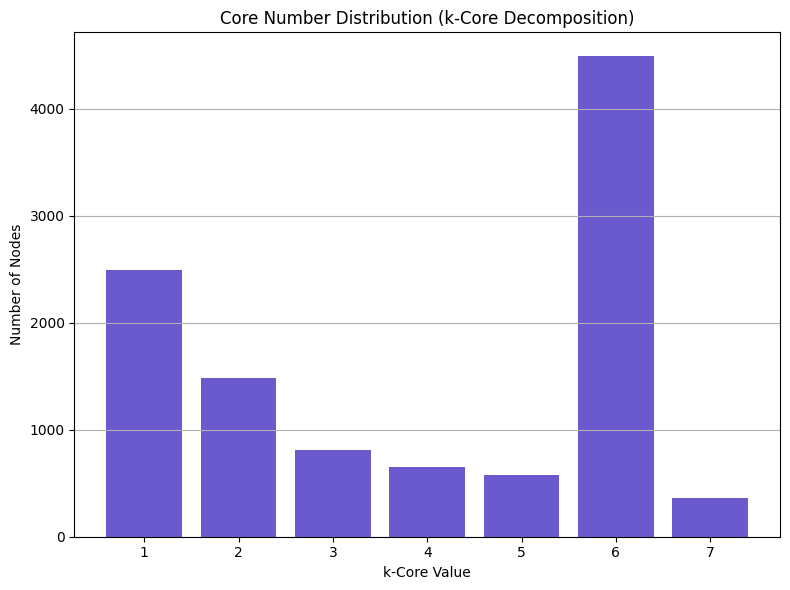
**Interpretation**: The modularity-based partitioning indicates weak global community structure, which aligns with the decentralized nature of peer-to-peer networks such as Gnutella.

**8. Core-Periphery & Path-Based Analysis *(Pending)***

To better understand structural depth and resilience:

**Core-Periphery (k-core Decomposition):**

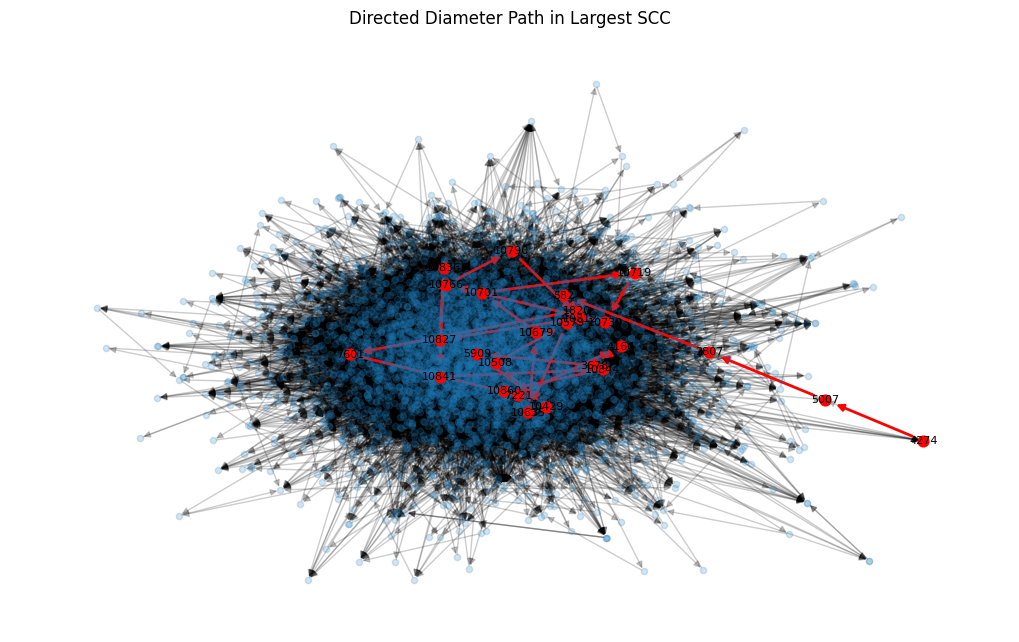
* The network was decomposed into **k-cores**, identifying a deep inner core where nodes are highly interconnected.
* Nodes in **high k-cores** represent the structural backbone of the network.
* Nodes in **lower k-cores** are more loosely attached and vulnerable to disconnection.

****

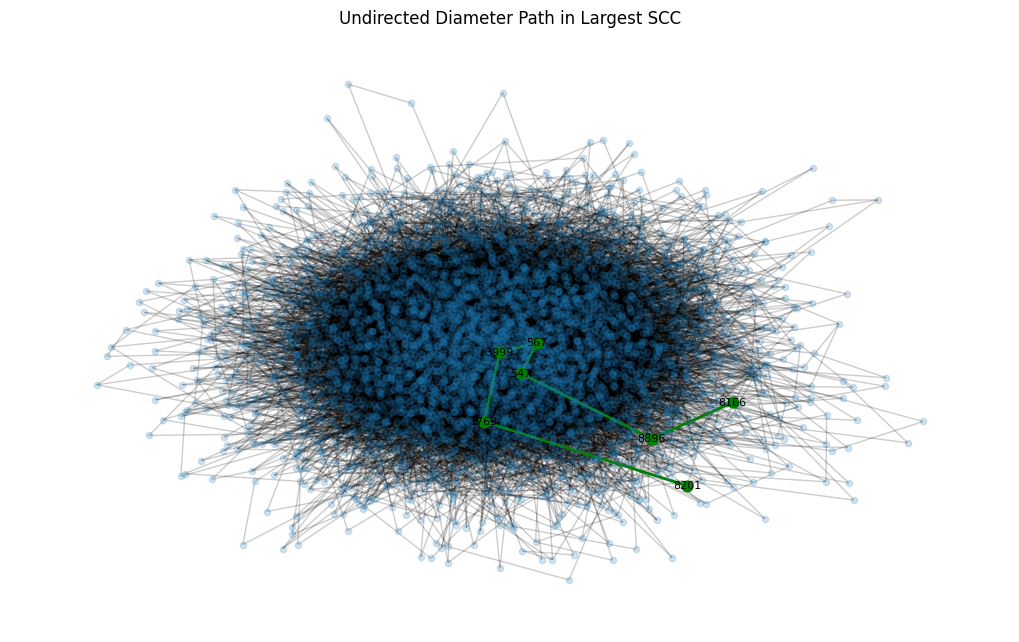
***Fig 5.*** *K-core visualization highlighting the central, densely connected core nodes.*

**Shortest Path & Diameter:**

* As the full graph is not strongly connected, we extracted the **largest strongly connected component (SCC)**.
* Within this SCC:
  + **Average shortest path length** represents typical hop distance between any two nodes.
  + **Diameter** indicates the longest minimal path between any node pair in the SCC.
* This gives a realistic insight into **functional reachability** and communication efficiency within the network’s active core.



***Fig 6.*** *Directed Diameter Path in Largest Strongly Connected Component*

****

***Fig 7.*** *Undirected Diameter Path in Largest Strongly Connected Component*

**9. Discussion**

From the results obtained so far:

* The Gnutella network is sparse, low in clustering, and disassortative.
* Centrality metrics reveal a small number of high-impact nodes likely super-peers or hub servers.
* The graph exhibits scale-free properties in its degree distribution, consistent with unregulated P2P networks.

Further interpretation regarding small-world phenomena, community strength, and core-periphery structure will be included after the pending sections are completed.

**10. Conclusion**

This report presented the initial analysis of the p2p-Gnutella04 network using NetworkX. We successfully extracted basic metrics, degree properties, and key centrality measures. Observed properties are consistent with typical peer-to-peer technological networks namely, sparse connectivity, low clustering levels, and the presence of high-degree hub nodes.

Future work will focus on:

* Community detection and modularity
* Core-periphery structure
* Shortest-path and diameter analysis
* Gephi-based visual exploration and comparative analysis

**11. References**

1. SNAP Datasets - <https://snap.stanford.edu/data/>
2. NetworkX Documentation - <https://networkx.org/>
3. Newman, M. E. (2010). Networks: an introduction.
4. Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, *2008*(10), P10008.
5. Barabási, A. L. (2002). The new science of networks. *Cambridge MA. Perseus*.

**Appendix**

**A. Figures**

1. Fig 1. Histogram of node degrees in the full Gnutella network.
2. Fig 2. Visualization of a 1000-node subgraph.
3. Fig 3. Community detection result in Gephi using the Louvain method with ForceAtlas2 layout.
4. Fig 4. Visualization of the largest community detected using NetworkX's greedy modularity algorithm.
5. Fig 5. K-core visualization highlighting the central, densely connected core nodes.
6. Fig 6. Directed Diameter Path in Largest Strongly Connected Component
7. Fig 7. Undirected Diameter Path in Largest Strongly Connected Component

**12.Code**

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

from networkx.algorithms.community import greedy\_modularity\_communities

input\_path = 'p2p-Gnutella04.txt'

output\_path = 'p2p-Gnutella04.csv'

df = pd.read\_csv(input\_path, sep='\t', comment='#', header=None, names=["FromNodeId", "ToNodeId"])

df.to\_csv(output\_path, index=False)

print("sucessfully converted.")

file\_path = "p2p-Gnutella04.txt"

df = pd.read\_csv(file\_path, sep="\t", comment='#', header=None, names=["FromNodeId", "ToNodeId"])

print("Dataset Overview:")

print("--------------------------------")

print(f"Total number of edges: {len(df)}")

unique\_nodes = pd.unique(df[["FromNodeId", "ToNodeId"]].values.ravel())

print(f"Total number of nodes: {len(unique\_nodes)}")

has\_weights = df.shape[1] > 2

print(f"Does the dataset include edge weights?: {'Yes' if has\_weights else 'No'}")

G = nx.DiGraph()

G.add\_edges\_from(df.values)

print("\n Graph Analysis:")

print("--------------------------------")

print(f"Is the graph directed?: {'Yes' if G.is\_directed() else 'No'}")

print(f"Graph density: {nx.density(G):.6f}")

print(f"Number of weakly connected components: {nx.number\_weakly\_connected\_components(G)}")

isolated\_nodes = list(nx.isolates(G))

print(f"Number of isolated nodes: {len(isolated\_nodes)}")

*# Degree analysis*

print("\nDegree Analysis:")

print("--------------------------------")

in\_degrees = dict(G.in\_degree())

out\_degrees = dict(G.out\_degree())

total\_degrees = dict(G.degree())

avg\_in = sum(in\_degrees.values()) / len(in\_degrees)

avg\_out = sum(out\_degrees.values()) / len(out\_degrees)

avg\_total = sum(total\_degrees.values()) / len(total\_degrees)

print(f"Average in-degree: {avg\_in:.2f}")

print(f"Maximum in degree: {max(in\_degrees.values())}")

print(f"Minimum in degree: {min(in\_degrees.values())}")

print("---------------")

print(f"Average out-degree: {avg\_out:.2f}")

print(f"Maximum out degree: {max(out\_degrees.values())}")

print(f"Minimum out degree: {min(out\_degrees.values())}")

print("---------------")

print(f"Average total degree: {avg\_total:.2f}")

print(f"Maximum total degree: {max(total\_degrees.values())}")

print(f"Minimum total degree: {min(total\_degrees.values())}")

top\_degrees = sorted(total\_degrees.items(), key=lambda x: x[1], reverse=True)[:3]

print("\nTop 3 nodes by total degree:")

for node, deg in top\_degrees:

    print(f"Node {node}: {deg} degree(s)")

from collections import Counter

*# Count frequency of each total degree*

degree\_values = list(total\_degrees.values())

degree\_freq = Counter(degree\_values)

top\_10\_degrees = degree\_freq.most\_common(10)

print("\nTop 10 most frequent total degrees:")

print("-----------------------------------")

for degree, count in top\_10\_degrees:

    print(f"Degree {degree}: {count} nodes")

degrees, counts = zip(\*top\_10\_degrees)

plt.figure(figsize=(8, 6))

plt.bar([str(d) for d in degrees], counts, color='darkcyan')

plt.title("Top 10 Most Frequent Total Degrees")

plt.xlabel("Degree")

plt.ylabel("Number of Nodes")

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

plt.figure(figsize=(10, 6))

plt.hist(in\_degrees.values(), bins=50, color='royalblue', edgecolor='black')

plt.title("In-Degree Distribution")

plt.xlabel("In-Degree")

plt.ylabel("Number of Nodes")

plt.grid(True)

plt.tight\_layout()

plt.show()

plt.figure(figsize=(10, 6))

plt.hist(out\_degrees.values(), bins=50, color='darkorange', edgecolor='black')

plt.title("Out-Degree Distribution")

plt.xlabel("Out-Degree")

plt.ylabel("Number of Nodes")

plt.grid(True)

plt.tight\_layout()

plt.show()

*# Compute HITS scores*

print("Running HITS Algorithm (ε = 1.0e-4)...")

hits\_hubs, hits\_authorities = nx.hits(G, max\_iter=1000, tol=1.0e-4, normalized=True)

*# Get top 10 hub scores*

top\_hubs = sorted(hits\_hubs.items(), key=lambda x: x[1], reverse=True)[:10]

top\_auths = sorted(hits\_authorities.items(), key=lambda x: x[1], reverse=True)[:10]

*# Display*

print("\nTop 10 Nodes by Hub Score:")

print("---------------------------")

for node, score in top\_hubs:

    print(f"Node {node}: Hub Score = {score:.5f}")

print("\nTop 10 Nodes by Authority Score:")

print("-------------------------------")

for node, score in top\_auths:

    print(f"Node {node}: Authority Score = {score:.5f}")

plt.figure(figsize=(10, 6))

plt.hist(hits\_authorities.values(), bins=50, color='darkorange', edgecolor='black')

plt.title("Authority Score Distribution")

plt.xlabel("Authority Score")

plt.ylabel("Number of Nodes")

plt.grid(True)

plt.tight\_layout()

plt.show()

sample\_nodes = list(G.nodes())[:100]

sample\_subgraph = G.subgraph(sample\_nodes)

*# Get hub scores for sampled nodes*

hub\_scores\_sample = [hits\_hubs.get(n, 0) for n in sample\_subgraph.nodes()]

*# Create layout*

pos = nx.spring\_layout(sample\_subgraph, seed=42)

*# Draw hub scores*

plt.figure(figsize=(10, 8))

nodes = nx.draw\_networkx\_nodes(

    sample\_subgraph, pos,

    node\_color=hub\_scores\_sample,

    cmap='Blues',

    node\_size=100

)

nx.draw\_networkx\_edges(sample\_subgraph, pos, alpha=0.4)

plt.title("Hub Score Visualization (Sample Subgraph)")

plt.colorbar(nodes, label="Hub Score")

plt.axis('off')

plt.show()

print("\nCentrality Measures:")

print("--------------------------------")

*# PageRank*

pagerank = nx.pagerank(G)

avg\_pagerank = sum(pagerank.values()) / len(pagerank)

top\_pagerank = sorted(pagerank.items(), key=lambda x: x[1], reverse=True)[:3]

print(f"Average PageRank: {avg\_pagerank:.5f}")

print("Top 3 nodes by PageRank:")

for node, score in top\_pagerank:

    print(f"Node {node}: PageRank = {score:.5f}")

*# Betweenness*

betweenness = nx.betweenness\_centrality(G)

avg\_betweenness = sum(betweenness.values()) / len(betweenness)

top\_betweenness = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:3]

print(f"\nAverage Betweenness Centrality: {avg\_betweenness:.5f}")

print("Top 3 nodes by Betweenness Centrality:")

for node, score in top\_betweenness:

    print(f"Node {node}: Betweenness = {score:.5f}")

*# Closeness*

closeness = nx.closeness\_centrality(G)

avg\_closeness = sum(closeness.values()) / len(closeness)

top\_closeness = sorted(closeness.items(), key=lambda x: x[1], reverse=True)[:3]

print(f"\nAverage Closeness Centrality: {avg\_closeness:.5f}")

print("Top 3 nodes by Closeness Centrality:")

for node, score in top\_closeness:

    print(f"Node {node}: Closeness = {score:.5f}")

*# top 10 page rank*

top10\_pagerank = sorted(pagerank.items(), key=lambda x: x[1], reverse=True)[:10]

nodes\_pr = [str(n) for n, \_ in top10\_pagerank]

scores\_pr = [s for \_, s in top10\_pagerank]

plt.figure(figsize=(10, 6))

plt.bar(nodes\_pr, scores\_pr, color='steelblue')

plt.title("Top 10 Nodes by PageRank")

plt.xlabel("Node ID")

plt.ylabel("PageRank Score")

plt.xticks(rotation=45)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

*# top 10 betweenness*

top10\_betweenness = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]

nodes\_bt = [str(n) for n, \_ in top10\_betweenness]

scores\_bt = [s for \_, s in top10\_betweenness]

plt.figure(figsize=(10, 6))

plt.bar(nodes\_bt, scores\_bt, color='mediumpurple')

plt.title("Top 10 Nodes by Betweenness Centrality")

plt.xlabel("Node ID")

plt.ylabel("Betweenness Score")

plt.xticks(rotation=45)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

*# top 10 closeness*

top10\_closeness = sorted(closeness.items(), key=lambda x: x[1], reverse=True)[:10]

nodes\_cl = [str(n) for n, \_ in top10\_closeness]

scores\_cl = [s for \_, s in top10\_closeness]

plt.figure(figsize=(10, 6))

plt.bar(nodes\_cl, scores\_cl, color='indianred')

plt.title("Top 10 Nodes by Closeness Centrality")

plt.xlabel("Node ID")

plt.ylabel("Closeness Score")

plt.xticks(rotation=45)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

print("\nClustering and Transitivity (Undirected & Directed):")

print("------------------------------------------------------")

undirected\_G = G.to\_undirected()

filtered\_nodes = [n for n in undirected\_G.nodes() if undirected\_G.degree(n) >= 2]

avg\_clustering\_undirected = nx.average\_clustering(undirected\_G, nodes=filtered\_nodes)

transitivity\_undirected = nx.transitivity(undirected\_G)

print(f"Undirected average clustering coefficient: {avg\_clustering\_undirected:.5f}")

print(f"Undirected transitivity: {transitivity\_undirected:.5f}")

*# Directed metrics*

transitivity\_directed = nx.transitivity(G)

print(f"Directed transitivity (triadic closure): {transitivity\_directed:.5f}")

*# Assortativity*

assortativity = nx.degree\_pearson\_correlation\_coefficient(G)

print(f"Assortativity coefficient (directed graph): {assortativity:.4f}")

print("\nVisualization (sample of 1000 nodes):")

print("--------------------------------")

sample\_nodes = list(G.nodes())[:1000]

subgraph = G.subgraph(sample\_nodes)

plt.figure(figsize=(10, 7))

nx.draw(subgraph, with\_labels=True, node\_size=50, arrows=True, edge\_color='gray')

plt.title("Sample Subgraph of 1000 Nodes")

plt.tight\_layout()

plt.show()

sample\_nodes = list(G.nodes())[:500]

sample\_subgraph = G.subgraph(sample\_nodes)

plt.figure(figsize=(12, 9))

nx.draw(sample\_subgraph, with\_labels=True, node\_size=80, arrows=True, alpha=0.7)

plt.title("Sample Subgraph Visualization (500 nodes)")

plt.show()

degrees = [deg for \_, deg in G.degree()]

plt.figure(figsize=(10, 6))

plt.hist(degrees, bins=100, color='skyblue')

plt.title("Degree Distribution")

plt.xlabel("Degree")

plt.ylabel("Frequency")

plt.grid(True)

plt.show()

in\_deg = dict(G.in\_degree())

out\_deg = dict(G.out\_degree())

plt.figure(figsize=(8, 6))

plt.scatter(list(in\_deg.values()), list(out\_deg.values()), alpha=0.5)

plt.xlabel("In-Degree")

plt.ylabel("Out-Degree")

plt.title("In-Degree vs Out-Degree")

plt.grid(True)

plt.show()

degrees = [deg for \_, deg in G.degree()]

plt.figure(figsize=(10, 6))

plt.hist(degrees, bins=100, color='skyblue', edgecolor='black')

plt.title("Degree Distribution (Total)")

plt.xlabel("Degree")

plt.ylabel("Number of Nodes")

plt.grid(True)

plt.show()

pagerank\_scores = nx.pagerank(G)

top\_nodes = sorted(pagerank\_scores.items(), key=lambda x: x[1], reverse=True)[:10] *#top nodes*

top\_node\_ids = [n for n, \_ in top\_nodes]

colors = ['red' if n in top\_node\_ids else 'lightblue' for n in sample\_subgraph.nodes()]

plt.figure(figsize=(12, 9))

nx.draw(sample\_subgraph, with\_labels=True, node\_color=colors, node\_size=80, arrows=True)

plt.title("Top PageRank Nodes (in red)")

plt.show()

print("\nComponent Size Analysis")

print("========================")

*# 1. Weakly Connected Components (WCC)*

wcc = list(nx.weakly\_connected\_components(G))

wcc\_sizes = [len(c) for c in wcc]

print(f"\n[Directed] Weakly Connected Components:")

print(f"→ Total: {len(wcc)}")

print(f"→ Largest component size: {max(wcc\_sizes)}")

print(f"→ Smallest component size: {min(wcc\_sizes)}")

plt.figure(figsize=(10, 5))

plt.hist(wcc\_sizes, bins=50, color='orange', edgecolor='black')

plt.title("Weakly Connected Component Sizes (Directed)")

plt.xlabel("Component Size")

plt.ylabel("Frequency")

plt.grid(True)

plt.tight\_layout()

plt.show()

*# 2. Strongly Connected Components (SCC)*

scc = list(nx.strongly\_connected\_components(G))

scc\_sizes = [len(c) for c in scc]

print(f"\n[Directed] Strongly Connected Components:")

print(f"→ Total: {len(scc)}")

print(f"→ Largest component size: {max(scc\_sizes)}")

print(f"→ Smallest component size: {min(scc\_sizes)}")

plt.figure(figsize=(10, 5))

plt.hist(scc\_sizes, bins=50, color='skyblue', edgecolor='black')

plt.title("Strongly Connected Component Sizes (Directed)")

plt.xlabel("Component Size")

plt.ylabel("Frequency")

plt.grid(True)

plt.tight\_layout()

plt.show()

*# 3. Connected Components (Undirected)*

undirected = G.to\_undirected()

ucc = list(nx.connected\_components(undirected))

ucc\_sizes = [len(c) for c in ucc]

print(f"\n[Undirected] Connected Components:")

print(f"→ Total: {len(ucc)}")

print(f"→ Largest component size: {max(ucc\_sizes)}")

print(f"→ Smallest component size: {min(ucc\_sizes)}")

plt.figure(figsize=(10, 5))

plt.hist(ucc\_sizes, bins=50, color='seagreen', edgecolor='black')

plt.title("Connected Component Sizes (Undirected)")

plt.xlabel("Component Size")

plt.ylabel("Frequency")

plt.grid(True)

plt.tight\_layout()

plt.show()

undirected = G.to\_undirected()

communities = list(greedy\_modularity\_communities(undirected))

print(f"Number of communities found: {len(communities)}")

largest = max(communities, key=len)

print(f"Size of the largest community: {len(largest)}")

subG = undirected.subgraph(largest)

node\_degrees = dict(subG.degree())

node\_colors = [node\_degrees[n] for n in subG.nodes()]

node\_sizes = [80 + 2 \* node\_degrees[n] for n in subG.nodes()]

pos = nx.kamada\_kawai\_layout(subG)

plt.figure(figsize=(12, 10))

nx.draw\_networkx\_nodes(subG, pos, node\_color=node\_colors, cmap=plt.cm.plasma, node\_size=node\_sizes)

nx.draw\_networkx\_edges(subG, pos, edge\_color='lightgray', alpha=0.5)

plt.title("Largest Detected Community (Node color = Degree)", fontsize=14)

plt.axis('off')

plt.tight\_layout()

plt.show()

import community as community\_louvain

*# Louvain method on undirected graph*

partition = community\_louvain.best\_partition(undirected)

*# Number of communities*

num\_communities = len(set(partition.values()))

print(f"Number of communities (Louvain): {num\_communities}")

community\_sizes = [len(c) for c in communities]

plt.figure(figsize=(10, 6))

plt.hist(community\_sizes, bins=30, color='mediumseagreen', edgecolor='black')

plt.title("Community Size Distribution")

plt.xlabel("Community Size (Number of Nodes)")

plt.ylabel("Number of Communities")

plt.grid(True)

plt.show()

triangle\_counts = nx.triangles(undirected)

total\_triangles = sum(triangle\_counts.values()) // 3

print(f"Total number of triangles (undirected): {total\_triangles}")

plt.figure(figsize=(10, 6))

plt.hist(triangle\_counts.values(), bins=30, color='orange', edgecolor='black')

plt.title("Triangle Count per Node (Undirected)")

plt.xlabel("Number of Triangles")

plt.ylabel("Number of Nodes")

plt.grid(True)

plt.show()

undirected = G.to\_undirected()

core\_numbers = nx.core\_number(undirected)

max\_core = max(core\_numbers.values())

print("Core-Periphery (k-Core) Analysis")

print("----------------------------------")

print(f"Maximum k-core value: {max\_core}")

*# Count how many nodes belong to each core level*

from collections import Counter

core\_distribution = Counter(core\_numbers.values())

print("\nNumber of nodes in each k-core:")

for k, count in sorted(core\_distribution.items()):

    print(f"k = {k}: {count} nodes")

*# Plot number of nodes per k-core*

ks, counts = zip(\*sorted(core\_distribution.items()))

plt.figure(figsize=(8, 6))

plt.bar(ks, counts, color='slateblue')

plt.title("Core Number Distribution (k-Core Decomposition)")

plt.xlabel("k-Core Value")

plt.ylabel("Number of Nodes")

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

largest\_scc\_nodes = max(nx.strongly\_connected\_components(G), key=len)

*# Directed & Undirected subgraphs*

largest\_scc\_digraph = G.subgraph(largest\_scc\_nodes).copy()

largest\_scc\_undirected = G.to\_undirected().subgraph(largest\_scc\_nodes).copy()

print("Path Length and Diameter Analysis (Corrected)")

print("---------------------------------------------")

print(f"Directed SCC size: {largest\_scc\_digraph.number\_of\_nodes()} nodes")

*# Directed average path length*

avg\_path\_length = nx.average\_shortest\_path\_length(largest\_scc\_digraph)

print(f"Average shortest path length (directed): {avg\_path\_length:.4f}")

*# Undirected diameter*

diameter\_undir = nx.diameter(largest\_scc\_undirected)

print(f"Diameter (undirected): {diameter\_undir}")

*# Directed diameter*

all\_pairs = dict(nx.all\_pairs\_shortest\_path\_length(largest\_scc\_digraph))

max\_dist = 0

diameter\_nodes\_directed = (None, None)

for source, targets in all\_pairs.items():

    for target, dist in targets.items():

        if dist > max\_dist:

            max\_dist = dist

            diameter\_nodes\_directed = (source, target)

print(f"Directed diameter: {max\_dist}")

print(f"Nodes in directed diameter path: {diameter\_nodes\_directed}")

path\_directed = nx.shortest\_path(largest\_scc\_digraph, source=diameter\_nodes\_directed[0], target=diameter\_nodes\_directed[1])

pos\_directed = nx.spring\_layout(largest\_scc\_digraph, seed=42)

plt.figure(figsize=(10, 6))

nx.draw(largest\_scc\_digraph, pos\_directed, node\_size=20, alpha=0.2, with\_labels=False)

nx.draw\_networkx\_nodes(largest\_scc\_digraph, pos\_directed, nodelist=path\_directed, node\_color='red', node\_size=60)

nx.draw\_networkx\_edges(largest\_scc\_digraph, pos\_directed, edgelist=list(zip(path\_directed, path\_directed[1:])), edge\_color='red', width=2)

nx.draw\_networkx\_labels(largest\_scc\_digraph, pos\_directed, labels={n: str(n) for n in path\_directed}, font\_size=8)

plt.title("Directed Diameter Path in Largest SCC")

plt.axis('off')

plt.tight\_layout()

plt.show()

diameter\_path\_undir = nx.diameter(largest\_scc\_undirected)  *# for size*

source, target = nx.periphery(largest\_scc\_undirected)[0], nx.periphery(largest\_scc\_undirected)[-1]

path\_undir = nx.shortest\_path(largest\_scc\_undirected, source=source, target=target)

pos\_undir = nx.spring\_layout(largest\_scc\_undirected, seed=42)

plt.figure(figsize=(10, 6))

nx.draw(largest\_scc\_undirected, pos\_undir, node\_size=20, alpha=0.2, with\_labels=False)

nx.draw\_networkx\_nodes(largest\_scc\_undirected, pos\_undir, nodelist=path\_undir, node\_color='green', node\_size=60)

nx.draw\_networkx\_edges(largest\_scc\_undirected, pos\_undir, edgelist=list(zip(path\_undir, path\_undir[1:])), edge\_color='green', width=2)

nx.draw\_networkx\_labels(largest\_scc\_undirected, pos\_undir, labels={n: str(n) for n in path\_undir}, font\_size=8)

plt.title("Undirected Diameter Path in Largest SCC")

plt.axis('off')

plt.tight\_layout()

plt.show()