

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

**2024-2025 ACADEMIC YEAR**

**SPRING SEMESTER**

CE479 – COMPLEX NETWORK ANALYSIS

**Course Supervisor:**

Asst. Prof. Dr. Sultan ZAVRAK

Structural and Centrality Analysis of the P2P-Gnutella04 Network

**Prepared by:**

Deniz KANLICALIOĞLU

Mustafa Mansur YÖNÜGÜL

**Student No:**

211015015

211015033

TABLE OF CONTENTS

**Page No**

ABSTRACT iii

1. INTRODUCTION 4

2. DATASET DESCRIPTION 4

3. METHODOLOGY (PYTHON – NETWORKX) 5

3.1. Graph Construction 5

3.2. Basic Graph Statistics 5

4. NETWORK ANALYSIS WITH NETWORKX 7

4.1. Degree Distribution 7

4.2. Centrality Measures 8

4.3. Clustering & Transitivity 8

4.4. Degree Assortativity 9

5. VISUALIZATION 9

6. GEPHI ANALYSIS 10

7. COMMUNITY DETECTION 13

8. CONCLUSION 17

9. REFERENCES 18

10. FIGURES 18

11. CODE 19

ABSTRACT

This study presents a comprehensive analysis of the p2p-Gnutella04 peer-to-peer file-sharing network using both programmatic (NetworkX) and visual (Gephi) approaches. The dataset, sourced from SNAP, comprises 10,876 nodes and 39,994 directed, unweighted edges. After importing the edge list via Pandas into a networkx.DiGraph, we conducted fundamental integrity checks, confirming no isolated nodes and a single weakly connected component despite 188 strongly connected components. Topological analysis revealed a sparse graph (density ≈ 0.000338) with a heavy-tailed degree distribution indicative of scale-free behavior. Centrality measures PageRank, betweenness, closeness, and HITS identified a small subset of high-impact hub nodes critical to network connectivity. Clustering and transitivity metrics were low (average clustering ≈ 0.008; transitivity ≈ 0.005), and degree assortativity was mildly negative (≈ –0.003), reflecting a technological network structure. Community detection using both greedy modularity (23 communities, largest ≈ 583 nodes) and Louvain methods in Gephi (29 communities) highlighted a fragmented modular architecture. k-core decomposition exposed a resilient inner backbone, while shortest-path and diameter analyses on the largest strongly connected component quantified communication efficiency (average path length ≈ 4.2 hops; diameter ≈ 15). Gephi’s ForceAtlas2 visualizations corroborated NetworkX findings and offered finer modular resolution. These insights demonstrate the power of combining algorithmic and exploratory tools to unravel the structural and functional patterns of large-scale networks.

# INTRODUCTION

As digital systems from social media platforms to biological data networks become increasingly complex, network science has emerged as a vital field for analyzing and understanding these interconnections. As noted by Newman, the study of complex networks has become a cornerstone in understanding systems ranging from social interactions to the Internet and biological processes [1].

This project aims to provide a practical understanding of complex network structures through the analysis of a large-scale real-world dataset, applying key concepts from network science using Python and NetworkX.

This study investigates the p2p-Gnutella04 dataset, which models the structure of a peer-to-peer file-sharing system recorded in 2004. The dataset, made available by the Stanford Network Analysis Project (SNAP), represents a directed graph structure and was explored using various computational methods. The analysis focuses on uncovering key topological patterns like degree distribution and clustering, along with evaluating centrality indicators such as PageRank and betweenness to highlight node influence within the network. Structural aspects, including component analysis and assortativity, are also examined in depth. It also includes community detection, core-periphery analysis, and visual exploration using Gephi to offer deeper insight into the modular and hierarchical organization of the network.

# DATASET DESCRIPTION

The dataset used in this study, known as p2p-Gnutella04, captures a snapshot of a peer-to-peer file-sharing network observed in August 2004. Each node in the network represents a Gnutella peer, and a directed edge from one node to another indicates a connection initiated from the source peer to the target. The network is unweighted and directed, comprising 10,876 nodes and 39,994 edges in total.

Originally provided as a plain text edge list without headers, the dataset was converted to .csv format to facilitate data manipulation using Python libraries such as Pandas and NetworkX. This transformation enabled more efficient graph construction and analysis.

The dataset is part of the Stanford Network Analysis Project (SNAP) and is publicly available at https://snap.stanford.edu/data/p2p-Gnutella04.html [2].

# METHODOLOGY (PYTHON – NETWORKX)

## Graph Construction

To begin the analysis, the raw edge list provided in the .txt format was loaded into a Pandas DataFrame. The dataset was then transformed into a directed graph object using NetworkX’s DiGraph() function, which allowed each peer-to-peer connection to be modeled with clear source and target directions.

Following the construction of the graph, basic structural checks were carried out. These checks includes verifying the graph’s directionality, identifying isolated nodes, and examining the number and size of connected components. The graph was found to contain no isolated nodes, and connectivity was further assessed by exploring both strongly and weakly connected components using NetworkX functions.

## Basic Graph Statistics

Several foundational metrics were calculated to better understand the general structure and density of the network:

* Graph Type: Directed
* Total Nodes: 10,876
* Total Edges: 39,994
* Graph Density: 0.000338
* Weakly Connected Components: 1
* Strongly Connected Components: 188
* Isolated Nodes: 0

The extremely low density highlights that this is a sparse network, where only a small fraction of all possible connections between nodes actually exist. Moreover, the existence of a single weakly connected component covering the entire graph indicates that all nodes are at least indirectly reachable, although strong mutual reachability is limited to smaller internal clusters. These characteristics are consistent with the structure of decentralized, real-world peer-to-peer networks like Gnutella.

# NETWORK ANALYSIS WITH NETWORKX

## Degree Distribution

One of the first steps in analyzing the network was to examine how many connections each node had. Since the dataset is directed, we looked at three types of degrees: how many edges come into a node (in-degree), how many go out (out-degree), and their total. On average, each node had about 3.68 incoming and outgoing links, which shows that most peers only connect to a small number of others.

To better visualize this, we plotted a histogram of all the total degree values. The result was a highly uneven distribution most nodes had just a few connections, while a handful had very large degrees. This kind of skewed pattern is typical in scale-free networks, where a small group of nodes often called "hubs" handle a large portion of the traffic and play a major role in keeping the network together.

A graph of a number of degrees

AI-generated content may be incorrect.

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

## Centrality Measures

To pinpoint the most influential or structurally significant nodes, multiple centrality measures were applied:

PageRank helped identify nodes with strong influence based on the volume and quality of their incoming links. Nodes with high PageRank scores were likely to function as authorities within the network.

Betweenness centrality revealed which nodes served as bridges between different parts of the graph by calculating how often a node appeared on the shortest paths between others. These nodes are vital for the efficient flow of information.

Closeness centrality assessed how accessible a node is to all others in the network. A high closeness score implies that a node can reach others quickly via relatively short paths, reflecting its positional efficiency within the system.

Each centrality perspective highlights different roles a node might play whether as an influencer, connector, or efficient communicator offering a multidimensional view of node importance in the Gnutella network.

## Clustering & Transitivity

To evaluate the network’s tendency to form tightly-knit groups, clustering metrics were computed. Since NetworkX’s clustering functions are designed for undirected graphs, the original directed structure was temporarily converted into an undirected format. This allowed for the application of standard clustering algorithms.

The average clustering coefficient calculated approximately 0.008, while the global transitivity which reflects the overall ratio of closed triplets to all possible triplets in the network was around 0.005. Both of these values are relatively low, indicating that the nodes in the Gnutella network rarely form triangles or local clusters. This outcome aligns with expectations for a technological or infrastructure-type network, where connections are driven by functionality rather than social affinity.

## Degree Assortativity

The concept of assortativity in network science refers to the tendency of nodes to connect with others that have similar characteristics in this case, degree. In the Gnutella network, the degree assortativity coefficient was found to be slightly negative (–0.003).

This negative value suggests a disassortative mixing pattern, meaning that high-degree nodes are more likely to be connected with low-degree ones. Such behavior is common in decentralized systems, particularly in communication networks, where a few central hubs serve many peripheral nodes. This structural imbalance enhances robustness but also implies a reliance on those core nodes for overall connectivity.

# VISUALIZATION

Due to the large size of the Gnutella network, directly visualizing the full graph was not practical. Instead, smaller portions of the network were selected and plotted to get a clearer view of the structure. These subgraphs containing 500 or 1,000 randomly chosen nodes allowed us to observe how peers are locally connected without overwhelming the visual layout.

To generate the visualizations, we used NetworkX along with Matplotlib. Nodes were positioned using layout algorithms like spring layout, which helps spread the nodes out and reduce clutter. In some versions, node size and color were adjusted based on properties such as degree, allowing more central nodes to stand out.

Although these plots do not reflect the entire network, they still provide a useful snapshot of the overall structure. They make it easier to identify local hubs and loosely connected clusters, offering a visual complement to the numerical analysis presented in previous sections.

**A computer generated image of a network

AI-generated content may be incorrect.**

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

# GEPHI ANALYSIS

NetworkX works well for calculating various network metrics, and less suited for visually exploring large graphs. To address this, Gephi was used as a complementary tool to interactively visualize the Gnutella network and verify the patterns found through Python-based analysis.

The data was first exported from Python into a .csv format, which was then imported into Gephi. Once loaded, the ForceAtlas2 layout algorithm applied to position the nodes. This algorithm dynamically organizes the graph such that nodes with stronger or more frequent connections are drawn closer together, naturally forming clusters without manual adjustment.

In order to reveal deeper structural patterns, node colors were assigned using the Louvain community detection algorithm. For this method it automatically grouped densely interconnected nodes into the same cluster. As a result, the visualization displayed 29 distinct communities, each represented with a different color, making the modular structure of the network easy to interpret.

In addition to coloring, node sizes were scaled based on degree, allowing highly connected nodes hubs to visually stand out. This further emphasized the presence of a small set of central nodes playing a dominant role in the network's connectivity.

Overall, Gephi provided a visually intuitive overview of the network’s modularity and supported the earlier findings from NetworkX. It was especially helpful for observing local community structures and understanding the layout of connections among peers in the Gnutella system.

**renklilik, yeşil, daire içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

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

The visualization above illustrates the modular structure of the Gnutella network using Gephi. Nodes are colored based on the Louvain community detection algorithm, and their sizes are scaled by degree centrality. As seen in the image, the network is dominated by a dense central region with several large hub nodes, visualized as oversized circles. These central nodes act as critical connection points, linking various peripheral peers and small communities. The outer regions of the graph are composed of low-degree nodes, many of which are only sparsely connected.

Visibly distinct color clusters indicates well defined community boundaries, where nodes within the same group are more connected to each other than to the rest of the network. This is a typical feature of peer-to-peer systems, where localized clusters of users interact heavily while still relying on a few major hubs for broader connectivity.

# COMMUNITY DETECTION

To explore the community structure within the Gnutella network, modularity-based clustering was carried out using NetworkX’s implementation of the Greedy Modularity algorithm. Since this algorithm only supports undirected graphs, the original directed network was temporarily converted into an undirected version for this step.

The detection process revealed 234 separate communities, reflecting the network’s fragmented nature. The largest community contained 583 nodes, which is still only a small fraction of the total network size. This suggests that while localized clusters exist, there is no dominant overarching community uniting a significant portion of the network.

The figure below highlights the structure of the largest community identified. Node sizes represent their degree, and node colors reflect degree variations as well, helping to distinguish more central nodes within the cluster.

*A purple and white circle with dots

AI-generated content may be incorrect.*

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

**Interpretation:**

These results points loosely modular structure, where many small communities coexist without forming large, cohesive blocks. It is a consistent pattern with the characteristics of peer-to-peer systems, which typically operate without a centralized structure. Instead of a few dominant groups, the network is composed of many modestly sized clusters that are weakly connected to each other. Observed modularity structure reinforces the idea that Gnutella’s topology favors decentralization and robustness over tightly-knit community formation.CORE-PERIPHERY & PATH-BASED ANALYSIS

To explore the network’s internal structure and how resilient it is to fragmentation, two key analyses were performed: k-core decomposition and path-based connectivity measurements.

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

The k-core decomposition technique was used to peel back layers of the network and identify its dense core. In this method, nodes are grouped based on how many connections they maintain with other well-connected nodes.

Nodes that belong to higher k-cores form the structural backbone of the network these are the nodes that remain even after removing all loosely connected peers. In contrast, those in lower k-shells tend to be more isolated and more vulnerable to removal, as their connectivity relies on a small number of links.

The visualization below shows the inner core of the network after applying k-core decomposition. Nodes in the highest cores are densely packed and form a concentrated region at the center, highlighting their importance in maintaining overall network cohesion.

**A graph of a number of numbers

AI-generated content may be incorrect.**

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

**Path-Based Analysis: Shortest Paths and Diameter**

Because the original graph is not strongly connected, the analysis focused on the largest strongly connected component (SCC) to better assess reachability within the network.

Inside this SCC, two key metrics were calculated:

The average shortest path length, which indicates the typical number of hops required for information to travel between any two nodes.

The diameter, which represents the longest shortest path between any node pair within the SCC.

These values provide insight into how efficiently the network can transmit data within its most connected region. The diameter also highlights the maximum communication delay that could occur in the worst-case scenario within this component.

Visualizations were created for both the directed and undirected versions of the SCC to offer a more complete view of path dynamics in different structural contexts.

A blue and red network

AI-generated content may be incorrect.

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

**A blue and green network

AI-generated content may be incorrect.**

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

# CONCLUSION

The analysis of the p2p-Gnutella04 network has revealed several key structural characteristics that align with the expectations of a decentralized peer-to-peer system. First and foremost, the network demonstrates a sparse topology, with a very low density and minimal clustering activity. This suggests that most nodes connect to only a few others, and tight-knit triadic structures are rare. The degree assortativity was found to be slightly negative, indicating a disassortative mixing pattern, where high-degree nodes tend to link with low-degree ones a pattern often seen in technological networks rather than social ones.

Centrality metrics such as PageRank, betweenness, and closeness revealed the presence of a small group of highly influential nodes. These nodes likely serve as super-peers or backbone hubs, enabling communication across otherwise loosely connected areas. Furthermore, the degree distribution followed a long-tailed pattern, consistent with scale-free network behavior, where a few nodes dominate in terms of connectivity.

The visual and algorithmic exploration conducted in Gephi further confirmed the modular and hub-oriented nature of the network. Clusters identified through Louvain community detection were relatively small and well-separated, reinforcing the idea that Gnutella does not rely on large cohesive groups but instead operates through many isolated or weakly connected clusters.

Core-periphery analysis highlighted a resilient inner core, where highly connected nodes play a critical role in holding the network together. At the same time, path-based metrics within the largest strongly connected component showed that communication is efficient within this active core—reflected in moderate average path lengths and diameter values.

**Summary:**

* In summary, the Gnutella network behaves as expected for a real-world, decentralized P2P system:
* Sparse connectivity and low clustering reflect unregulated growth.
* Few central hubs ensure stability and efficiency.
* Community structure is weak but present.
* Core-periphery hierarchy and reasonable path efficiency demonstrate robustness.

# REFERENCES

[1] Newman, Mark, Networks: An Introduction, 1st edn (Oxford, 2010; online edn, Oxford Academic, 1 Sept. 2010), https://doi.org/10.1093/acprof:oso/9780199206650.001.0001, accessed 16 May 2025.

[2] Jure, L. (2014). SNAP Datasets: Stanford large network dataset collection. Retrieved December 2021 from <http://snap.stanford.edu/data>.

[3] 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.

[4] Barabási, A. L. (2002). The new science of networks. Cambridge MA. Perseus.

# FIGURES

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

Fig 2. Visualization of a 1000-node subgraph.

Fig 3. Community detection result in Gephi using the Louvain method with ForceAtlas2 layout.

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

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

Fig 6. Directed Diameter Path in Largest Strongly Connected Component

Fig 7. Undirected Diameter Path in Largest Strongly Connected Component

# CODE

#!/usr/bin/env python  
# coding: utf-8  
  
# In[3]:  
  
  
import pandas as pd  
import networkx as nx  
import matplotlib.pyplot as plt  
from networkx.algorithms.community import greedy\_modularity\_communities  
  
  
# In[4]:  
  
  
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.")  
  
  
# ## Network Dataset Analysis  
#   
# This code snippet analyzes a P2P network dataset (`p2p-Gnutella04.txt`):  
#   
# ### Key Metrics:  
# - \*\*Edges\*\*: Counts directed connections (e.g., `A→B`)   
# - \*\*Nodes\*\*: Identifies unique entities in the network   
# - \*\*Weights\*\*: Checks if connections have numeric intensities   
#   
  
# In[5]:  
  
  
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'}")  
  
  
  
  
# ## Graph Structure Analysis  
#   
# This code analyzes the topological properties of the constructed directed graph:  
#   
# ### Key Metrics:  
# - \*\*Directed\*\*: Checks if edges are one-way   
# - \*\*Density\*\*: Ratio of actual edges to possible edges (sparsity measure)   
# - \*\*Weakly Connected Components\*\*: Subgraphs connected when ignoring edge direction   
# - \*\*Isolated Nodes\*\*: Nodes with no connections   
  
# In[6]:  
  
  
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)}")  
  
  
  
# This code examines node connectivity patterns in the directed graph:  
#   
# ### Key Metrics:  
# - \*\*In-degree\*\*: Number of incoming edges (popularity measure)  
# - \*\*Out-degree\*\*: Number of outgoing edges (activity measure)   
# - \*\*Total degree\*\*: Sum of in/out edges (overall connectivity)  
  
# In[8]:  
  
  
# 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)")  
  
  
# In[9]:  
  
  
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")  
  
# Bar chart  
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()  
  
  
# In[10]:  
  
  
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()  
  
  
# In[11]:  
  
  
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()  
  
  
# ## Centrality Measures in Network Graph  
#   
# This section analyzes the centrality of nodes in the graph `G` using three commonly used metrics:  
#   
# ### 1. PageRank Centrality  
# - \*\*Definition:\*\* PageRank is used to rank nodes based on their importance, taking into account both the number and quality of links to a node.  
# - \*\*Code Steps:\*\*  
# - `nx.pagerank(G)`: Calculates PageRank for all nodes in the graph.  
# - `avg\_pagerank`: Computes the average PageRank value.  
# - The top 3 nodes are extracted based on their PageRank scores and printed.  
#   
# ### 2. Betweenness Centrality  
# - \*\*Definition:\*\* Betweenness measures how often a node appears on the shortest paths between pairs of nodes. High betweenness suggests a node is a bridge or bottleneck.  
# - \*\*Code Steps:\*\*  
# - `nx.betweenness\_centrality(G)`: Calculates betweenness centrality for each node.  
# - `avg\_betweenness`: Average of all betweenness values.  
# - Displays the top 3 nodes with highest betweenness values.  
#   
# ### 3. Closeness Centrality  
# - \*\*Definition:\*\* Closeness indicates how close a node is to all other nodes in the network. A higher score means shorter distances on average.  
# - \*\*Code Steps:\*\*  
# - `nx.closeness\_centrality(G)`: Computes closeness centrality.  
# - `avg\_closeness`: Mean of all closeness values.  
# - Prints the top 3 nodes by closeness.  
  
# ### HITS Metric Summary  
#   
# The HITS algorithm was applied with a convergence threshold of ε = 1.0e-4. Results show:  
#   
# - \*\*Hub nodes\*\*: Act as good linkers, pointing to authoritative nodes.  
# - \*\*Authority nodes\*\*: Are considered trustworthy or popular (frequently referenced).  
# - The score distributions are skewed, with a small number of dominant nodes — typical of scale-free networks like P2P topologies.  
#   
# This analysis complements PageRank by separating the concepts of \*influence\* and \*relevance\*.  
#   
  
# In[15]:  
  
  
# 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()  
  
  
# In[12]:  
  
  
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}")  
  
  
# In[ ]:  
  
  
# 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()  
  
  
# In[ ]:  
  
  
# 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()  
  
  
# In[ ]:  
  
  
# 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()  
  
  
# ## Clustering, Transitivity, and Assortativity Analysis  
#   
# This section focuses on measuring how nodes cluster together and how similar nodes tend to connect in the graph `G`.  
#   
# ### 1. Average Clustering Coefficient  
# - \*\*Definition:\*\* The clustering coefficient of a node quantifies how close its neighbors are to being a complete clique (fully connected).  
# - \*\*Calculation:\*\*   
# - `nx.average\_clustering(undirected\_G)` computes the mean clustering coefficient over all nodes in the \*\*undirected\*\* version of the graph.  
# - \*\*Insight:\*\* Higher values indicate a more tightly-knit local neighborhood structure.  
#   
# ### 2. Transitivity  
# - \*\*Definition:\*\* A global form of the clustering coefficient. It measures the probability that adjacent nodes of a node are connected.  
# - \*\*Calculation:\*\*   
# - `nx.transitivity(undirected\_G)` calculates the ratio of triangles to triplets in the graph.  
# - \*\*Insight:\*\* Indicates how interconnected the network is at a global scale.  
#   
# ### 3. Assortativity Coefficient  
# - \*\*Definition:\*\* Measures the similarity of connections in the graph with respect to the node degree.  
# - \*\*Calculation:\*\*  
# - `nx.degree\_pearson\_correlation\_coefficient(G)` computes the Pearson correlation coefficient of node degrees at either end of an edge.  
# - \*\*Insight:\*\*  
# - A \*\*positive\*\* value means nodes tend to connect to others with similar degree (assortative mixing).  
# - A \*\*negative\*\* value means high-degree nodes connect to low-degree ones (disassortative mixing).  
  
# In[ ]:  
  
  
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}")  
  
  
# NetworkX does not implement directed clustering coefficient natively.  
#   
# It falls back to G.to\_undirected() when using nx.average\_clustering(G)  
#   
# This is why your result is ~0.006 → It’s not truly the directed clustering  
#   
  
# In[10]:  
  
  
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()  
  
  
# In[11]:  
  
  
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()  
  
  
# ## Degree Distribution Analysis  
#   
# The degree distribution provides insight into how connections (edges) are distributed among nodes in the network.  
#   
# ### Plot Description  
# - \*\*X-axis (Degree):\*\* The number of edges connected to each node (node degree).  
# - \*\*Y-axis (Frequency):\*\* The number of nodes that have a given degree.  
#   
  
# In[12]:  
  
  
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-Degree vs Out-Degree Analysis  
#   
# This scatter plot compares the \*\*in-degree\*\* and \*\*out-degree\*\* of each node in the directed graph `G`.  
  
# In[13]:  
  
  
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()  
  
  
# In[ ]:  
  
  
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()  
  
  
# In[15]:  
  
  
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()  
  
  
# ## Weakly Connected Component Size Distribution  
#   
# This plot displays the size distribution of weakly connected components in the directed graph `G`.  
#   
# ### What is a Weakly Connected Component?  
# - A \*\*weakly connected component\*\* in a directed graph is a subgraph where each node is connected to every other node \*\*if direction is ignored\*\*.  
# - This means you can reach any node from any other \*\*when treating all edges as undirected\*\*.  
#   
# ### Code Summary  
  
# In[18]:  
  
  
import matplotlib.pyplot as plt  
import networkx as nx  
  
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()  
  
  
# By default:  
#   
# Gephi excludes isolated nodes from visual/connected component statistics unless specifically included.  
#   
# NetworkX includes all nodes — even isolated ones — when computing components.  
#   
# This could mean:  
#   
# Gephi considers only the giant component + a minor one, excluding any isolated single nodes.  
#   
# NetworkX sees one giant component, plus many trivial components of size 1 (the isolated nodes).  
#   
#   
  
# ## Community Detection using Greedy Modularity  
#   
# This section applies community detection on the undirected version of the graph `G` to uncover groups of densely connected nodes (communities).  
#   
# ### What is Community Detection?  
# - Community detection partitions the graph into subsets (communities) where nodes within the same group are more densely connected to each other than to the rest of the graph.  
# - This is useful for identifying \*\*functional modules\*\*, \*\*interest groups\*\*, or \*\*attack clusters\*\*, depending on the network domain.  
#   
# ### Method Used: Greedy Modularity  
# - `greedy\_modularity\_communities()` detects communities by optimizing \*\*modularity\*\*, a measure of the strength of division of a network into modules.  
# - \*\*Modularity\*\* evaluates the density of links inside communities compared to links between communities.  
  
# In[19]:  
  
  
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()  
  
  
# In[20]:  
  
  
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}")  
  
  
# Note: The number of communities found using NetworkX (20)(23) is lower than in Gephi (29),   
# because NetworkX’s `greedy\_modularity\_communities` algorithm produces fewer, larger communities.  
# Gephi uses the \*\*Louvain method\*\*, which detects smaller and more modular communities due to higher resolution and iterative optimization.  
  
# ### Community Size Distribution Interpretation  
#   
# The histogram shows the distribution of community sizes in the network. Most communities are small, while a few are significantly larger.  
#   
# This suggests a \*\*modular structure with many small clusters\*\* and a few dominant groups — consistent with the behavior of P2P networks where nodes cluster around content, location, or uptime.  
  
# In[21]:  
  
  
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()  
  
  
# ### Undirected Triangle Interpretation  
#   
# - A \*\*directed triangle\*\* (3-cycle) indicates a fully reciprocal feedback structure among three nodes.  
# - The number of \*\*3-cycles\*\* in this peer-to-peer network is relatively low — indicating limited mutual awareness or circular routing.  
# - The \*\*transitivity\*\* metric offers a global measure of triangle density relative to connected triplets — even in a directed context.  
  
# In[22]:  
  
  
triangle\_counts = nx.triangles(undirected)  
total\_triangles = sum(triangle\_counts.values()) // 3  
print(f"Total number of triangles (undirected): {total\_triangles}")  
  
# 4. Histogram of triangle counts per node  
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()  
  
  
# ### Core-Periphery (k-Core) Interpretation  
#   
# The maximum k-core value shows the deepest level of cohesion in the network.   
# Nodes in higher k-cores are part of the \*\*core\*\* — densely interconnected and resilient.  
# Nodes in lower k-cores form the \*\*periphery\*\*, typically less connected and more vulnerable to disconnection.  
# This analysis reveals the \*\*inner backbone\*\* of the network.  
  
# In[23]:  
  
  
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")  
  
  
# In[24]:  
  
  
# 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()  
  
  
# ### Shortest Paths and Diameter (on Largest SCC)  
#   
# Since the full graph and even the largest WCC are not strongly connected, we used the \*\*largest strongly connected component\*\* (SCC) for shortest-path calculations.  
#   
# - The \*\*average shortest path length\*\* shows typical communication cost (in hops) within this tightly connected cluster.  
# - The \*\*diameter\*\* represents the furthest distance between any two nodes in this component.  
#   
# This gives a realistic picture of connectivity and efficiency inside the most functionally active portion of the network.  
#   
  
# In[38]:  
  
  
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}")  
  
  
# In[39]:  
  
  
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()  
  
  
# In[40]:  
  
  
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()