ASTRO PHYSICS) COLLABORATION NETWORK

INTRO SOCIAL NETWORK PROJECT

DR.REEM ESSAM

ENG. FADY MAGED

OUR TEAM

Ali Mohamed Sayed Ahmed Ali	20221449583
Menna Allah Mohamed Abdelhamid Shweel	20221400420
Doaa Khamis Kamal	20221400409
Marwa Mohamed Atya	20221370533
Manar Mohamed Younis Ahmed	20221370636
Mayar Mohamed Abdelfattah	20221381232

Table of Contents

0.0 ARXIV ASTRO-PH ASTRO PHYSICS COLLABORATION NETWORK

- The Astro-Physics collaboration network from the arXiv repository represents a fascinating dataset that captures the structure of scientific collaborations in the field of astrophysics. This dataset is constructed from the e-print submissions to the Astro-Physics (ASTRO-PH) category on arXiv and reflects the co-authorship relationships between researchers over a decade-long period, from January 1993 to April 2003.
- In this network, nodes represent authors, and an undirected edge exists between two nodes if the corresponding authors have collaborated on a paper. For papers with multiple co-authors, the collaboration forms a fully connected subgraph, illustrating the interconnected nature of academic teamwork in this field.
-

INTRODUCTION

Problems Description

1.1 ∅ The program focuses on analyzing the Astro-Physics collaboration network by leveraging graph theory and network analysis techniques. It processes the dataset, represented as an edge list, to perform comprehensive exploration and analysis. The key objectives are as follows:

1.2

1. Compute Node Degree and Average Degree
   - Measure the connectivity of individual nodes and calculate the overall average degree to understand the general level of collaboration across the network.
2. Explore Graph Representations:
   - Represent the graph using various formats such as an edge list, adjacency list, and adjacency matrix, showcasing the versatility of network data representation and its impact on computation.
3. Calculate Centrality Measures:
   - Evaluate the importance of nodes within the network using metrics such as betweenness, closeness, and eigenvector centrality, helping to identify influential authors or groups.
4. Analyze Clustering Coefficients:
   - Calculate the clustering coefficient for individual nodes and the entire network
5. Determine Shortest Paths:
   - Compute the shortest paths between nodes to analyze the network's efficiency in terms of information flow and collaboration reach, providing insights into how closely connected the authors are.
6. Community Detection and Visualization

CODE EXPLANATION:

2.1 Import necessary libraries & Loading the Dataset

- NetworkX: powerful library for the creation, manipulation, and analysis of complex networks and graphs
- networkx.algorithms.community: used to identify clusters of closely collaborating authors.
- Then loads the dataset from the edge list using the networkx library. It computes and displays basic properties of the graph, such as the number of nodes, number of edges, and the average degree of the network

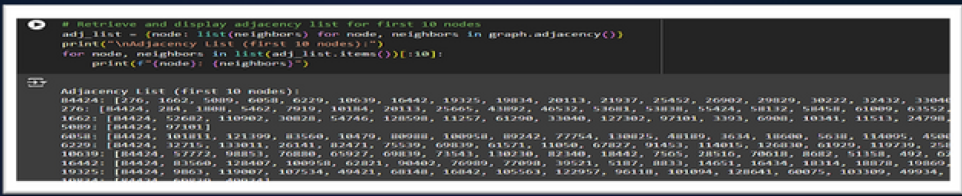
```
[188] # Calculate and display basic graph properties
      total_nodes = graph.number_of_nodes()
      total_edges = graph.number_of_edges()
      avg_degree = sum(dict(graph.degree()).values()) / total_nodes

[189] print(f"Graph Summary:")
      print(f"Nodes: {total_nodes}")
      print(f"Edges: {total_edges}")
      print(f"Average Degree: {avg_degree:.2f}")

      Graph Summary:
      Nodes: 18772
      Edges: 198110
      Average Degree: 21.11

[190] # Check if the graph is connected
      is_connected = nx.is_connected(graph)
      print("\nIs the graph connected?", is_connected, "\n")

      Is the graph connected? False

[191] # Check if graph is directed
      if nx.is_directed(graph):
          print("The graph is directed.")
      else:
          print("The graph is undirected.")

      The graph is undirected.
```

2.2 Graph Representations: Edge List, Adjacency List, and Adjacency Matrix

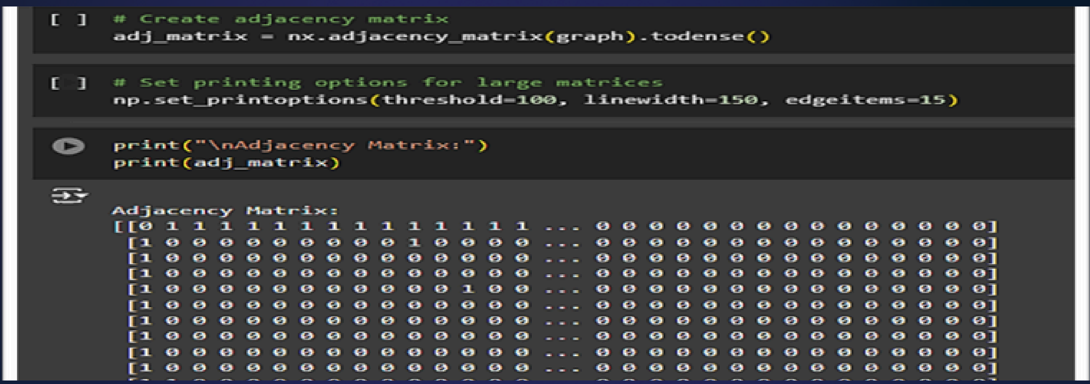
- The program explores various ways to represent the Astro-Physics collaboration network, each offering unique insights and computational advantages. Below are the details of these representations

1. Edge List:
Each row represents a connection between two nodes (authors).

2. Adjacency List:
The adjacency list represents the graph as a mapping where each node is associated with a list of its direct neighbours.



3. Adjacency Matrix:
The adjacency matrix is a two-dimensional array where each cell indicates the presence (or absence) of an edge between two nodes.



Centrality Measures

2.3

- The program computes three key centrality measures to evaluate the importance and influence of nodes within the collaboration network.
- the lowest centrality values may represent isolated or less influential parts of the network.
- But first take a subset from graph for easy visualization and analysis

```python
# Select a random subset of nodes for visualization
num_sample_nodes = 500
sample_nodes = list(graph.nodes())[:num_sample_nodes]
subgraph = graph.subgraph(sample_nodes)
```

1. Betweenness Centrality:

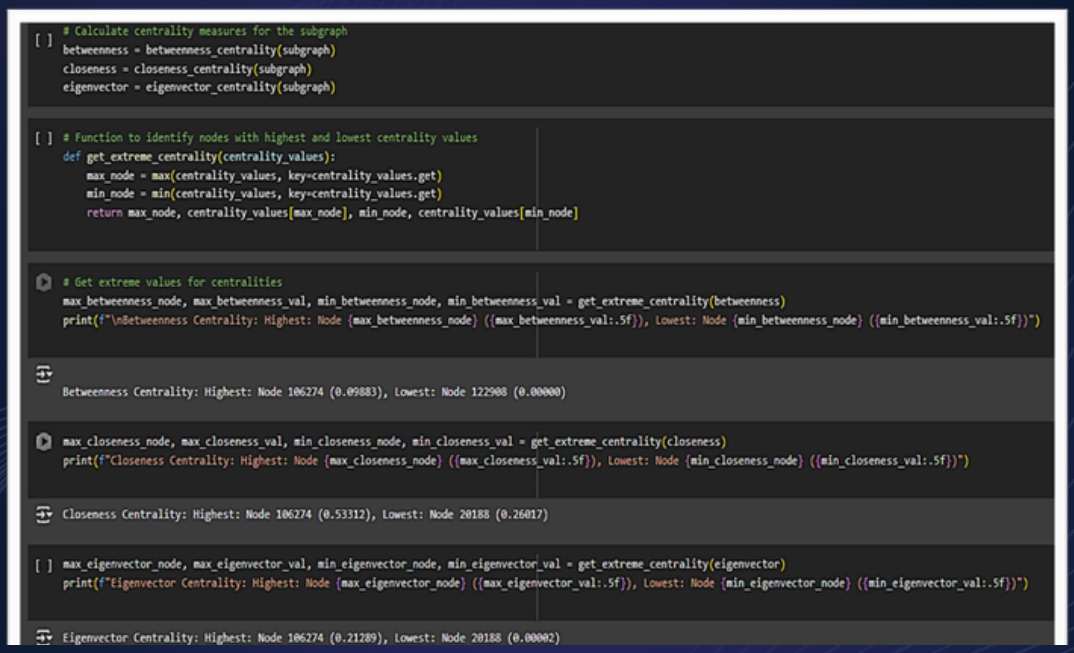
- Measures how often an author acts as a bridge between other pairs of authors in the network.
- Authors with high betweenness centrality often play the role of intermediaries, facilitating collaboration across different research groups or communities.

2. Closeness Centrality:

- Evaluates how close an author is to all other authors in the network, based on the average shortest path.
- Authors with high closeness centrality can quickly connect with others in the network

3. Eigenvector Centrality:

- Assesses an author's influence by considering their connections to other influential authors.
- Authors with high eigenvector centrality are deeply embedded within the network, often collaborating with other prominent or active researchers



```python
# Calculate centrality measures for the subgraph
betweenness = betweenness_centrality(subgraph)
closeness = closeness_centrality(subgraph)
eigenvector = eigenvector_centrality(subgraph)
```

```python
# Function to identify nodes with highest and lowest centrality values
def get_extreme_centrality(centrality_values):
    max_node = max(centrality_values, key=centrality_values.get)
    min_node = min(centrality_values, key=centrality_values.get)
    return max_node, centrality_values[max_node], min_node, centrality_values[min_node]
```

```python
# Get extreme values for centralities
max_betweenness_node, max_betweenness_val, min_betweenness_node, min_betweenness_val = get_extreme_centrality(betweenness)
print(f"\nBetweenness Centrality: Highest: Node {max_betweenness_node} ({max_betweenness_val:.5f}), Lowest: Node {min_betweenness_node} ({min_betweenness_val:.5f})")
```

```
Betweenness Centrality: Highest: Node 106274 (0.09883), Lowest: Node 122908 (0.00000)
```

```python
max_closeness_node, max_closeness_val, min_closeness_node, min_closeness_val = get_extreme_centrality(closeness)
print(f"Closeness Centrality: Highest: Node {max_closeness_node} ({max_closeness_val:.5f}), Lowest: Node {min_closeness_node} ({min_closeness_val:.5f})")
```

```
Closeness Centrality: Highest: Node 106274 (0.53312), Lowest: Node 20188 (0.26017)
```

```python
max_eigenvector_node, max_eigenvector_val, min_eigenvector_node, min_eigenvector_val = get_extreme_centrality(eigenvector)
print(f"Eigenvector Centrality: Highest: Node {max_eigenvector_node} ({max_eigenvector_val:.5f}), Lowest: Node {min_eigenvector_node} ({min_eigenvector_val:.5f})")
```

```
Eigenvector Centrality: Highest: Node 106274 (0.21289), Lowest: Node 20188 (0.00002)
```

DataFrame to display the results as a table

DataFrame to display the results as a table

```
# Create a DataFrame to display the results as a table
table = pd.DataFrame({
    "Node": list(subgraph.nodes),
    "Edge Count": [subgraph.degree[node] for node in subgraph.nodes],
    "Degree Centrality": [degree_centrality[node] for node in subgraph.nodes],
    "Betweenness Centrality": [betweenness[node] for node in subgraph.nodes],
    "Closeness Centrality": [closeness[node] for node in subgraph.nodes],
    "Eigenvector Centrality": [eigenvector[node] for node in subgraph.nodes],
    "Community": [community_mapping[node] for node in subgraph.nodes]
})

# Sort the table by Degree Centrality (or any other centrality measure if preferred)
table = table.sort_values(by="Degree Centrality", ascending=False).head(10)

# Print the table
print("\nNode Centrality Table:")
print(table.to_string(index=False))
```

```
Node Centrality Table:
  Node  Edge Count  Degree Centrality  Betweenness Centrality  Closeness Centrality  Eigenvector Centrality  Community
106274         185           0.370741                0.098830              0.533120                0.212894          0
 35290         156           0.312625                0.047842              0.520334                0.193979          0
 60471         149           0.298597                0.069931              0.507630                0.170141          0
100394         117           0.234469                0.029707              0.495040                0.161995          0
 39696         115           0.230461                0.022432              0.487305                0.160176          0
 19383         107           0.214429                0.009497              0.471645                0.155734          0
 67410         106           0.212425                0.009160              0.467666                0.160663          0
 55999          98           0.196393                0.009259              0.472091                0.147910          0
 36907          96           0.192385                0.015763              0.481196                0.090810          0
 47856          96           0.192385                0.020571              0.478887                0.091550          0
```

Community Detection in Collaboration Networks

2.4

- The program applies community detection techniques to uncover cohesive groups within the collaboration network, revealing clusters of authors who frequently collaborate. This is achieved using the Girvan-Newman algorithm, a well-known method for identifying communities by analyzing the structure of connections in the graph.

- ∅ Girvan-Newman Algorithm:
  - The Girvan-Newman algorithm identifies communities by iteratively removing edges with the highest betweenness centrality. These edges are considered critical links between different groups. As these edges are removed, the graph gradually splits into smaller subgraphs, each representing a community.
  - Nodes are color-coded based on their community membership, making the groups visually distinguishable.
  - Process:
    1) Compute betweenness centrality for all edges in the graph.
    2) Remove the edge with the highest betweenness centrality.
    3) Repeat until the graph is divided into distinct communities.

- But there are more algorithms such as:
  - Small Graphs: Girvan-Newman, Spectral Clustering.
  - Large Graphs: Louvain, Infomap, Label Propagation.
  - Dynamic Graphs: Label Propagation, Infomap.
  - Overlapping Communities:K-Clique Percolation, Infomap.

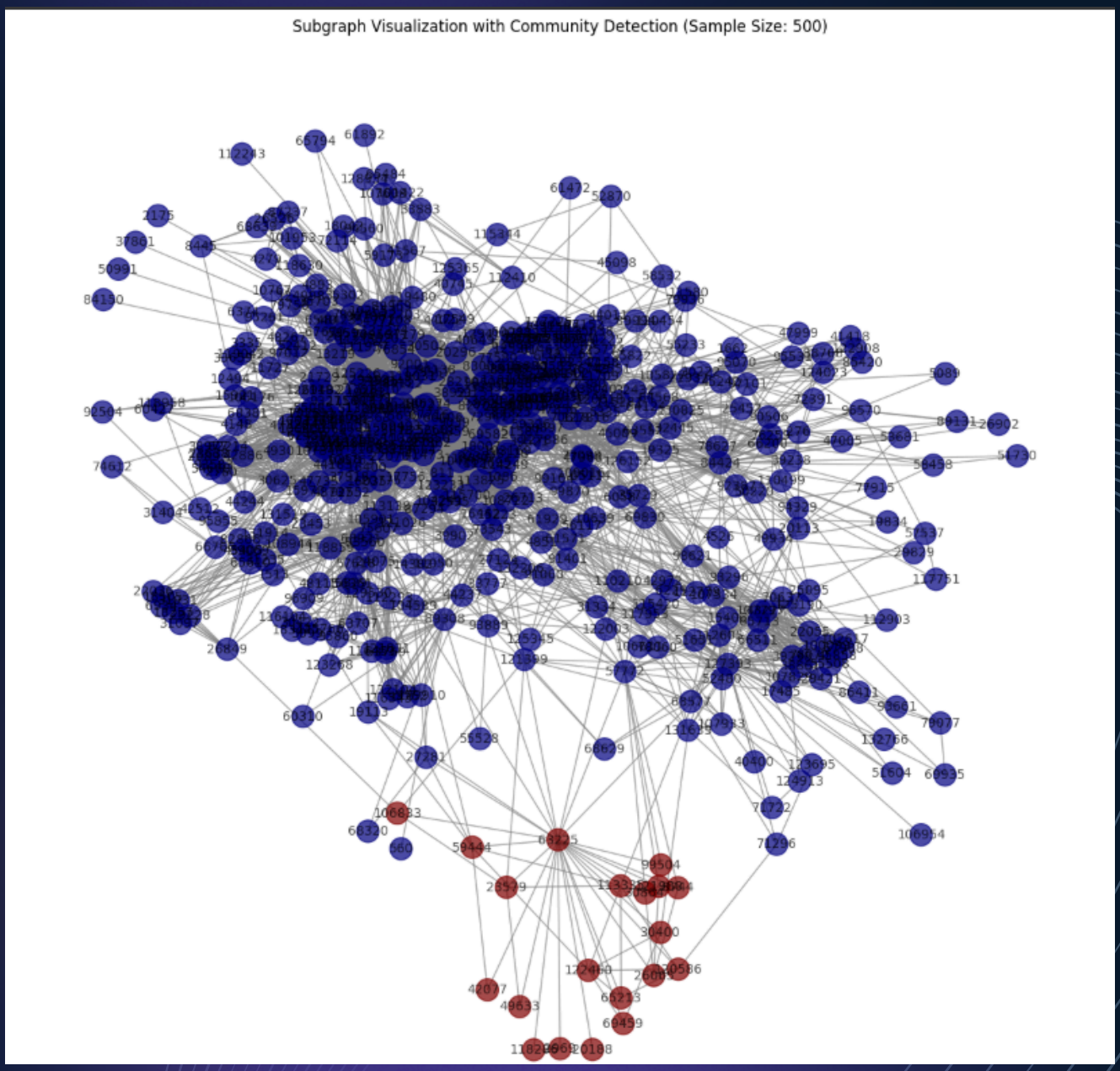
```
# Perform community detection using Girvan-Newman method
communities_gen = community.girvan_newman(subgraph)
first_level_communities = next(communities_gen)
community_mapping = {node: i for i, community in enumerate(first_level_communities) for node in community}

# Visualize the graph with community color coding
plt.figure(figsize=(12, 12))
layout = nx.spring_layout(subgraph)
node_colors = [community_mapping[node] for node in subgraph.nodes()]
nx.draw(subgraph, layout, node_color=node_colors, with_labels=True, node_size=50, cmap=plt.cm.jet, font_size=12)
plt.title(f"Subgraph Visualization with Community Detection (Sample Size: {num_sample_nodes})")
plt.show()
```

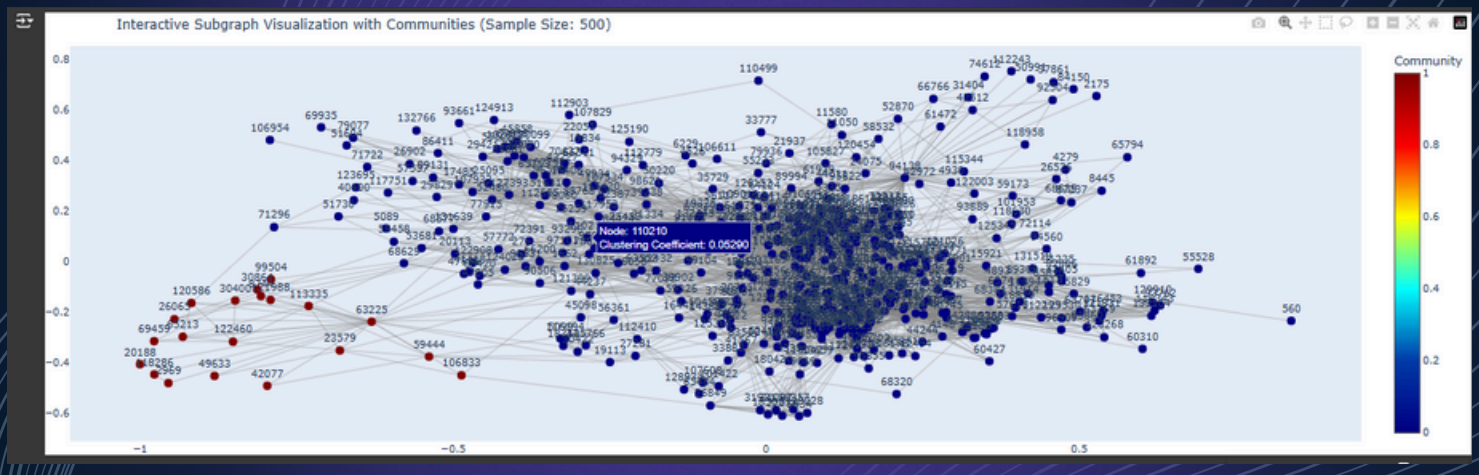
Visualization of interactive graph for The network

```
[255] import plotly.graph_objects as go
      def visualize_graph_with_plotly(subgraph, community_mapping, title="Graph with Communities"):
          """
          Visualizes a graph with Plotly, coloring nodes based on community membership.

          Parameters:
              subgraph: NetworkX Graph object (subset of nodes)
              community_mapping: Dictionary mapping node to community ID
              title: Title of the plot
          """
          # Compute layout for nodes
          pos = nx.spring_layout(subgraph, seed=42)

          # Extract edges and node positions
          edge_x = []     ## x-coordinates of edges
          edge_y = []    ### y-coordinates of edges
          for edge in subgraph.edges():
            # Get positions for the start and end of the edge
              x0, y0 = pos[edge[0]]   ### Start node
              x1, y1 = pos[edge[1]]  #### end node
               # Add coordinates for the edge trace (including None to break the line between edges)
              edge_x.extend([x0, x1, None])
              edge_y.extend([y0, y1, None])

      # Create the edge trace (lines connecting nodes)
          edge_trace = go.Scatter(
              x=edge_x,
              y=edge_y,
              line=dict(width=0.5, color='#888'),
              hoverinfo='none',
              mode='lines'
          )

          # Create node traces
          node_x = []  ## x-coordinates of nodes
          node_y = []   # y-coordinates of nodes
          node_color = []   ### Color of each node based on community
          node_labels = []
          node_hover_text = []  # Hover text for each node
```

```
[255]     for node in subgraph.nodes():
                  # Get node position
              x, y = pos[node]
              node_x.append(x)
              node_y.append(y)
              node_color.append(community_mapping[node])  # Community-based color
              node_labels.append(str(node))      # Assign label as the node identifier
               # Add hover info: Node ID and Clustering Coefficient
              hover_text = f"Node: {node}<br>Clustering Coefficient: {clustering_coefficients[node]:.5f}"
              node_hover_text.append(hover_text)


          node_trace = go.Scatter(
              x=node_x,
              y=node_y,
              mode='markers+text',      # Display markers (nodes) with text (labels)
              text=node_labels,     # Add node labels
              textposition="top center",      # Position labels above nodes
              hoverinfo='text',   # Show labels on hover
              hovertext=node_hover_text,  # Assign hover text

              marker=dict(
                  size=10,
                  color=node_color,
                  colorscale='Jet',  # Use Jet colormap
                  colorbar=dict(title="Community"),   # Add a colorbar with a title
                  showscale=True    # Show the color scale
              )
          )

          # Combine traces into a figure
          fig = go.Figure(data=[edge_trace, node_trace],
                          layout=go.Layout(
                              title=title,
                              titlefont_size=16,
                              showlegend=False,
                              hovermode='closest',

                              margin=dict(b=0, l=0, r=0, t=40),  # Adjust margins
                              xaxis=dict(showgrid=False, zeroline=False),
                              yaxis=dict(showgrid=False, zeroline=False)
                          ))
          fig.show()



      # Visualize the graph with Plotly
      visualize_graph_with_plotly(subgraph, community_mapping, f"Interactive Subgraph Visualization with Communities (Sample Size: {num_sample_nod
```

Visualization of graph for The network





Some Analysis (Degree distribution, Clustering Coefficient, Shortest path)

2.5

1. Clustering Coefficient:

measures how closely related authors are within their immediate network of co-authors. It provides a quantitative way to evaluate the degree of collaboration and the potential for creating tight-knit research communities or groups of authors that frequently collaborate with one another. A high clustering coefficient in this type of network indicates that collaborators tend to form densely connected groups, where most of an author's co-authors are also collaborators with each other

```
# Clustering Coefficient
clustering_coefficients = nx.clustering(graph)
avg_clustering = np.mean(list(clustering_coefficients.values()))

print(f"\nAverage Clustering Coefficient of the graph: {avg_clustering:.5f}")
```

```
Average Clustering Coefficient of the graph: 0.63059
```

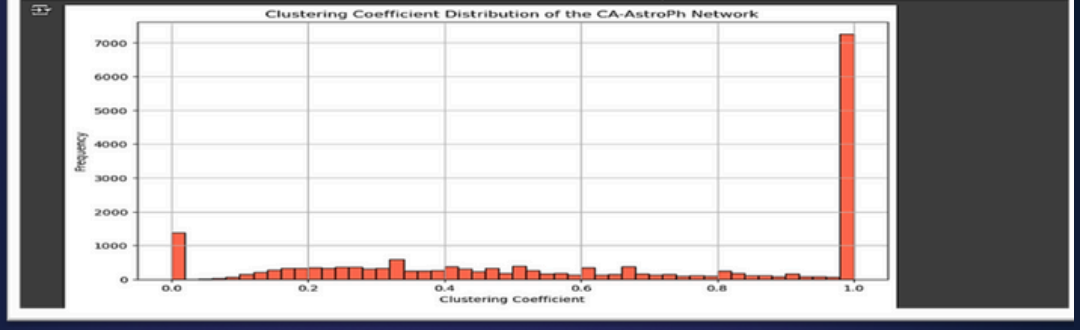
```
# Display Clustering Coefficients for the first few nodes
print("\nClustering Coefficients for the first few nodes:")
for node in list(clustering_coefficients.keys())[:10]:
    print(f"Node {node}: Clustering Coefficient = {clustering_coefficients[node]:.5f}")

# Convert clustering coefficient dictionary to DataFrame for better readability
clustering_df = pd.DataFrame(list(clustering_coefficients.items()), columns=["Node", "Clustering Coefficient"])

# Display the DataFrame
print("\nClustering Coefficients for All Nodes:")
print(clustering_df)
```

```
Clustering Coefficients for the first few nodes:
Node 84424: Clustering Coefficient = 0.07568
Node 276: Clustering Coefficient = 0.11613
Node 1662: Clustering Coefficient = 0.24142
Node 5089: Clustering Coefficient = 1.00000
Node 6058: Clustering Coefficient = 0.22921
Node 6229: Clustering Coefficient = 0.12427
Node 10639: Clustering Coefficient = 0.11879
Node 16442: Clustering Coefficient = 0.31494
Node 19325: Clustering Coefficient = 0.13085
Node 19834: Clustering Coefficient = 1.00000

Clustering Coefficients for All Nodes:
          Node  Clustering Coefficient
0        84424                0.075676
1          276                0.116129
2         1662                0.241417
3         5089                1.000000
4         6058                0.229213
...        ...                     ...
18767    32673                1.000000
18768    40403                0.000000
18769    36753                0.000000
18770    38713                0.000000
18771   129085                0.000000
```

```
# Nodes with highest and lowest clustering coefficient
max_clust_node = max(clustering_coefficients, key=clustering_coefficients.get)
min_clust_node = min(clustering_coefficients, key=clustering_coefficients.get)
print(f"\nNode with highest Clustering Coefficient: Node {max_clust_node} ({clustering_coefficients[max_clust_node]:.5f})")
print(f"Node with lowest Clustering Coefficient: Node {min_clust_node} ({clustering_coefficients[min_clust_node]:.5f})")
```

```
Node with highest Clustering Coefficient: Node 5089 (1.00000)
Node with lowest Clustering Coefficient: Node 37861 (0.00000)
```

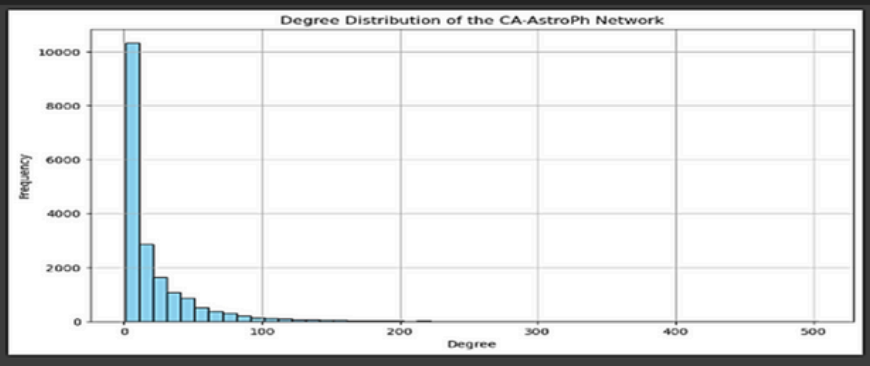
```
# Plotting Clustering Coefficient Distribution
plt.figure(figsize=(10, 6))
plt.hist(list(clustering_coefficients.values()), bins=50, color='tomato', edgecolor='black')
plt.title("Clustering Coefficient Distribution of the CA-AstroPh Network")
plt.xlabel("Clustering Coefficient")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



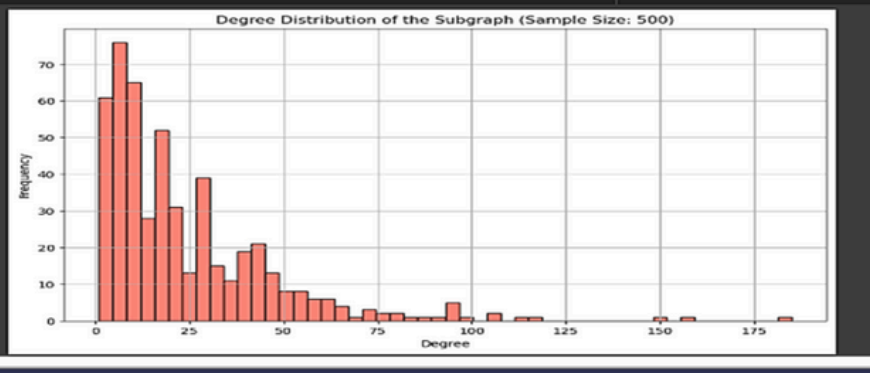
Node with highest Clustering Coefficient: Node 5089 (1.00000)
Node with lowest Clustering Coefficient: Node 37861 (0.00000)

Degree distribution

```python
# New plot: Degree distribution of the whole graph
degree_sequence = [deg for node, deg in graph.degree()]
plt.figure(figsize=(10, 6))
plt.hist(degree_sequence, bins=50, color='skyblue', edgecolor='black')
plt.title("Degree Distribution of the CA-AstroPh Network")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```




```python
# New plot: Degree distribution for the subgraph
subgraph_degree_sequence = [deg for node, deg in subgraph.degree()]
plt.figure(figsize=(10, 6))
plt.hist(subgraph_degree_sequence, bins=50, color='salmon', edgecolor='black')
plt.title("Degree Distribution of the Subgraph (Sample Size: 500)")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



Shortest path

```python
# New analysis: Shortest Path between two random nodes
node1, node2 = np.random.choice(list(graph.nodes()), size=2, replace=False)
shortest_path = nx.shortest_path(graph, source=node1, target=node2)

print(f"\nShortest Path between Node {node1} and Node {node2}: {shortest_path}")
```

```
Shortest Path between Node 103853 and Node 39367: [103853, 19736, 9039, 39037, 39367]
```