VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

REPORT

Social Network Analytics

Community Detection Techniques on Social Network

Name: Aditya Panditrao

Registration No.: 22MCB0032

Course Name: Social Network Analytics

Course Code: MCSE618L

INTRODUCTION

Social networks are related to health. Social network analysis has emerged as one tool for evaluating and understanding relationships between individual health and social environments, as well as developing interventions to prevent disease and improve public health. A variety of social network analytic methods are available to describe network composition, examine relationships between social networks and health, or identify the underlying group structure of social networks. Identifying groups within social networks is a highly relevant but underutilized tool for public health and prevention research. Typically, due to the diversity of ties in real-world networks, groups within a network are unclear. Community detection, the process of identifying subgroups of highly connected individual within a network, is one popular class of methods. Community detection has been used to examine many public health topics including HIV/AIDS, latrine ownership, smoking cessation, physical activity, cancer treatment patterns, and hospital service regions. Within a social network, communities are groups of individuals who are densely connected to each other, but have few connections to other individuals in the network. Once identified, community structures can help researchers examine if health tends to cluster within communities, or deliver targeted interventions to high-risk communities. Targeting communities is potentially more appealing than targeting specific individuals because individuals are often more amenable to change when the whole group changes at once, or some interventions may be more naturally targeted towards groups rather than individuals.

There are many community detection methods, or algorithms, available to researchers who are interested in social networks. Algorithms rely on different heuristics to place individuals into mutually exclusive groups. Applying these algorithms is complicated, mainly due to ambiguity around which algorithm is best for a given circumstance. Recent articles have attempted to provide guidance algorithm selection using criteria such as the mixing parameter of a network, computation time, or overlap with a simulated community structure. Other work has noted that the optimal community detection method will depend on how the communities will be used.Indeed, Yang et al specifically caution that guidance using criteria like computation time "have to be applied in conjunction with… research questions. A pure application of the recommendations could bias the results." This appears to be generally acknowledged by community detection researchers, but concrete demonstrations of how algorithms could be applied in conjunction with health promotion and disease prevention research questions are lacking.

Using a research question to drive algorithm choice allows researchers to leverage each Algorithm's different properties to produce a result that best aligns with their question. This Paper provides guidance to help prevention researchers with a background in network Analysis align their research questions with publicly available community detection algorithms. This method, the Question-Alignment (QA) approach, is intended to help Researchers align a research question with the most theoretically appropriate community Detection algorithm. This approach may not necessarily lead to better results but is designed to help researchers identify the most theoretically appropriate algorithm for their research question. This method requires a clearly defined research question and an understanding of the specific heuristics employed by algorithms. To help researchers apply the QA approach, the paper provides: an overview of six publicly available algorithms, (2) a discussion of research topics relevant to each algorithm, and (3) an applied example of the QA approach using real-world data.

Algorithms Overview

To begin, overviews for several definitions necessary to understand the described community detection algorithms are provided. Modularity quantifies the density of links within communities compared to links between communities (ranges from −1 to 1).16 Path lengths (or 'walks' between two nodes) are the number of edges one would have to use to 'walk' from one node to another. The shortest path length is the path between two nodes that involves traveling down the minimal number of edges. A random walk is a path between two nodes where each step is randomly chosen. Dendrograms are tree diagrams returned by some algorithms that visualize community divisions at each step of the algorithm. Six popular algorithms are considered. These algorithms are readily available in the R graph package, have shown good computational performance in networks of less than 1000 nodes, and have distinct features: Edge-Betweenness, Random Walktrap, Label Propagation, Infomap, Louvain, and Spinglass. Table 1 provides a high-level overview of each algorithm and outlines features that researchers may be interested in, such as their ability to handle directed networks or be customized. Algorithms are classified as divisive, agglomerative, or optimization based. Further information on each algorithm is available in the Appendix.

Divisive

Divisive algorithms begin with a complete network and iteratively divide the network into smaller communities. The Edge-Betweenness algorithm is one divisive method that defines communities by iteratively removing edges that have a high likelihood of linking separate communities. This algorithm continues until the full network has been completely divided (i.e., each node is its own community) and returns a dendrogram.

Agglomerative

Agglomerative algorithms begin by considering each node as its own community and then iteratively combine nodes into larger communities. The Walktrap method is based on the premise that nodes within communities are likely to be connected by shorter random "walks".28 Communities are iteratively merged together by minimizing the overall distance between nodes and communities, defined by random walks. The maximum path length of the random walks can be specified to result in more close-knit or more diverse communities. The algorithm continues until all nodes in the network are merged into a single community and returns a dendrogram. Label Propagation identifies communities through the transmission of labels between nodes. Each node starts with a unique label, and randomly selected nodes adopt the label of the majority of its neighbors. The algorithm stops when every node has the same label as the majority of adjacent nodes.

Optimization based

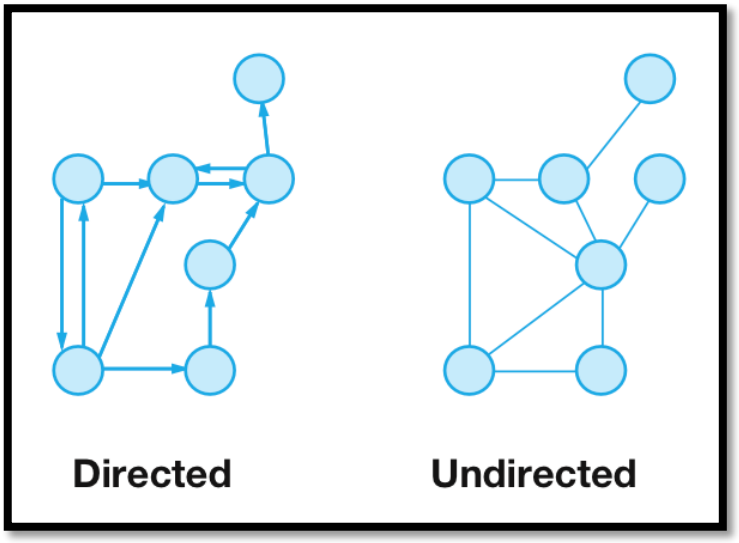
The main feature of optimization algorithms is finding the optimal solution for a prespecified objective function. The global optimum for modularity can be calculated using cluster_optimal() in the R igraph package. Due to the complexity of finding the global optimum for moderately sized networks (100+ nodes), other algorithms, which we focus on here, seek to more efficiently come to a solution. The Louvain algorithm, also known as Multilevel algorithm, is an efficient algorithm that seeks to maximize modularity by merging nodes into communities. The algorithm stops when no merges result in a modularity increase. The Spinglass algorithm optimizes a function that rewards edges with a community and penalizes edges between communities and stops when that function is minimized. That function can be modified to place a bigger (or smaller) emphasis on edges within communities. Finally, the Infomap algorithm focuses on optimizing the flow of information throughout a network.Communities are iteratively merged together to optimize information flow and the algorithm stops when no further optimization is possible.

Community detection in a network identifies and groups the more densely interconnected nodes in a given graph. This graph can take the form of a social network graph, a biological network, or a representation of a local network of computers, for example. Clusters of related nodes can be grouped using various algorithms. In this article, eight such algorithms are tested against three datasets, and their performance is evaluated.

Graphs

A graph is a nonlinear data structure that models pairwise relations between objects. A graph comprises nodes or vertices which are connected by edges or links.

An undirected graph consists of symmetric links, whereas a directed graph includes asymmetric, 'to and from' links between vertices. Undirected links can be traversed in any direction, whereas directed links can only be traversed in the direction of the link, from the origin to the destination node. A weighted graph is a graph in which links are assigned certain numeric values known as weights. Depending on the graph, these weights may represent costs, lengths, or other quantities.



An example of an undirected graph is the roads interconnecting cities that can be traversed in either direction. The relevant directed graph would be airline routes connecting the same fixed in a particular direction.

Graphs can model many types of relations and processes in scientific information systems. Graphs can be used to model and solve many practical problems.

Network Theory and Graphs

Graphs are thus versatile tools to represent various models, and in particular, networks. Network theory studies graphs as a representation of either symmetric or asymmetric relations between discrete objects. A network can be defined as a graph in which nodes and links have attributes like names and where the optional weight attribute represents a real-world quantity. The analysis of networks lends itself to various scientific fields like biology, physics, statistics, computer science, sociology, and statistics, to derive meaningful inferences from a data store.

Social network analysis, in particular, has emerged as a general application of community detection algorithms. With the rise of the World Wide Web and social networking sites, characterizing consequent communities yields a wealth of information about human behavioral patterns. Thus, there is a need for algorithms that detect such communities so that further meaningful inferences can be made from them.

Community

A community is defined as a subset of nodes within a graph where the connections or linkages are denser than the rest of the network. A network is said to have a community structure if the nodes of the network can be easily grouped into potentially overlapping sets of nodes, such that each set of nodes is densely connected internally.
Communities hold significance in a network because they provide insight into its overall entity. Taking the example of a biological application, in metabolic networks, communities correspond to cycles or pathways. In contrast, in protein interaction networks, communities correspond to proteins with similar functionality inside a biological cell.

In social networks, the existence of communities generally affects various processes like false news or epidemic spreading. To completely understand and carry out studies on these processes, community detection techniques are of the utmost importance.

Community Detection Algorithms

Community detection in a network is the process of identifying and grouping the more densely interconnected nodes in a given graph.

Clustering coefficient as a graph metric

The clustering coefficient of a graph is a measure of the degree to which nodes in a graph tend to cluster together. Evidence suggests that in most real-world networks, and in particular social networks, nodes tend to create tightly knit communities characterized by a relatively high density of links. This likelihood tends to be greater than the average probability of a link randomly established between two nodes. The clustering coefficient has therefore been used on each network that has been tested in this article as a means of getting an idea about its relative inter-connectivity.

Modularity as a CDA metric

Modularity measures the structure of a given network or graph when community detection is carried out. It measures the strength of the division of a network into modules (communities). Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules. The concept of modularity is used to evaluate the partition of a network into communities based on the intuition that random graph structures should not follow a community structure.

LFR benchmark for CDA evaluation

When it comes to detecting communities, there is still no universal agreement on what an ideal result of the community detection process looks like and by extension, the reliability of community detection algorithms. This is where a simple network model, the planted ℓ-partition model, comes into play. In this model, one ‘plants’ a partition in a graph, which consists of a certain number of groups of nodes.
The LFR benchmark is an example of a planted ℓ-partition model. It improves upon the previous GN benchmark by introducing power law distributions of degree and community size. There is also no restriction on the expected degrees or community sizes. LFR benchmark graphs can be generated relatively quickly, even for extensive networks. Thus, it has an advantage over the GN benchmark and is a better test for the performance of any given algorithm. In this article, a network generated by the LFR benchmark has been used to compare the algorithms.

DATASET USED

DBLP collaboration network and ground-truth communities

The DBLP computer science bibliography provides a comprehensive list of research papers in computer science. We construct a co-authorship network where two authors are connected if they publish at least one paper together. Publication venue, e.g, journal or conference, defines an individual ground-truth community; authors who published to a certain journal or conference form a community.
We regard each connected component in a group as a separate ground-truth community. We remove the ground-truth communities which have less than 3 nodes. We also provide the top 5,000 communities with highest quality which are described in our paper. As for the network, we provide the largest connected component.

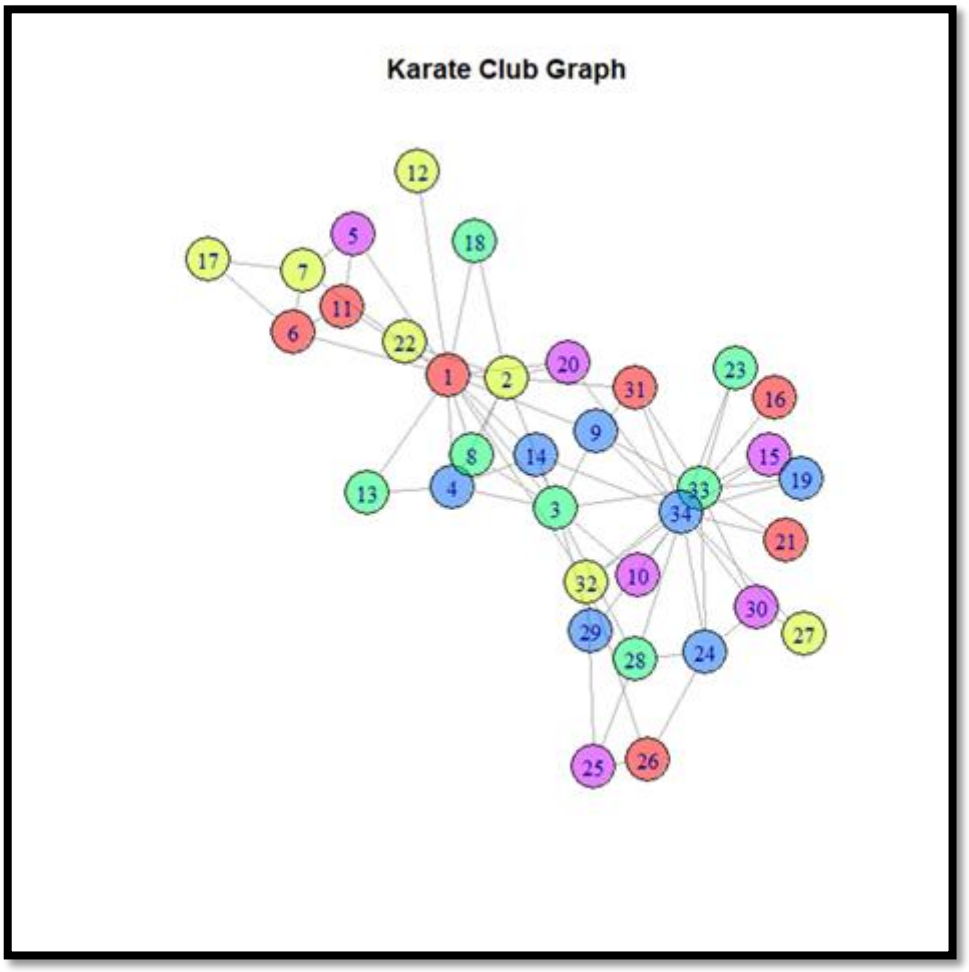
Dataset statistics	
Nodes	317080
Edges	1049866
Nodes in largest WCC	317080 (1.000)
Edges in largest WCC	1049866 (1.000)
Nodes in largest SCC	317080 (1.000)
Edges in largest SCC	1049866 (1.000)
Average clustering coefficient	0.6324
Number of triangles	2224385
Fraction of closed triangles	0.1283
Diameter (longest shortest path)	21
90-percentile effective diameter	8

Networks used

The choice of datasets has been made to show the varying impact of the input parameters on the output. A regular real-world network, an artificial network used as a benchmark, and a random graph have been used to test the working of all the algorithms used fairly.

Zachary's Karate Club network

In 1977 Wayne Zachary collected data about the members of a university karate club. Each node represents a club member, and each edge represents a tie between two members. This is a real-world undirected network with 34 nodes (members) and 78 edges. During this study, a fight arose between members of the club, leading to its splitting into two groups. An interesting problem that thus arose was figuring out these two groups by examining the nature of the links with the aid of community detection algorithms. Thus, this is an apt network to perform analysis and the results would be notable.



Girvan-Newman algorithm

The **Girvan-Newman algorithm** for the detection and analysis of community structure relies on the iterative elimination of edges that have the highest number of shortest paths between nodes passing through them. By removing edges from the graph one-by-one, the network breaks down into smaller pieces, so-called communities. The algorithm was introduced by Michelle Girvan and Mark Newman.

How does it work?

The idea was to find which edges in a network occur most frequently between other pairs of nodes by finding edges betweenness centrality. The edges joining communities are then expected to have a high edge betweenness. The underlying community structure of the network will be much more fine-grained once the edges with the highest betweenness are eliminated which means that communities will be much easier to spot.

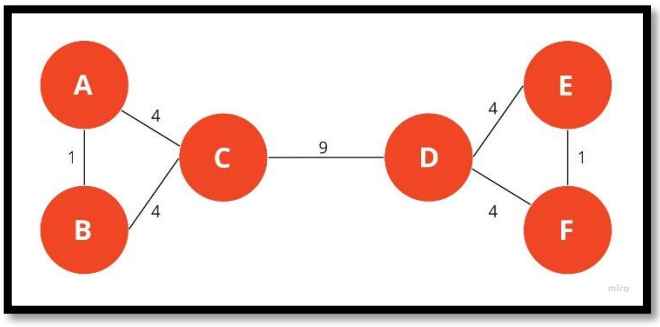
The Girvan-Newman algorithm can be divided into four main steps:

For every edge in a graph, calculate the edge betweenness centrality.

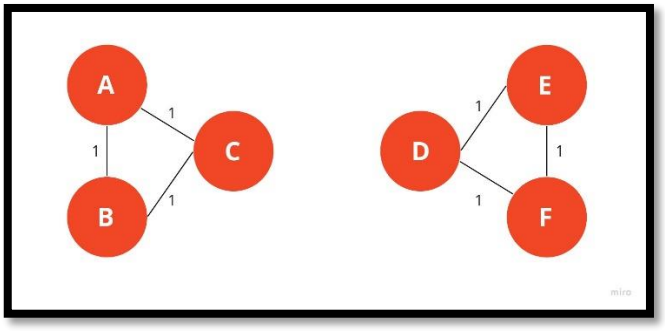
Remove the edge with the highest betweenness centrality.

Calculate the betweenness centrality for every remaining edge.

Repeat steps 2-4 until there are no more edges left.



In this example, you can see how a typical graph looks like when edges are assigned weights based on the number of shortest paths passing through them**.** To keep things simple, we only calculated the number of undirected shortest paths that pass through an edge. The edge between nodes **A** and **B** has a strength of 1 because we don't count **A->B** and **B->A** as two different paths.

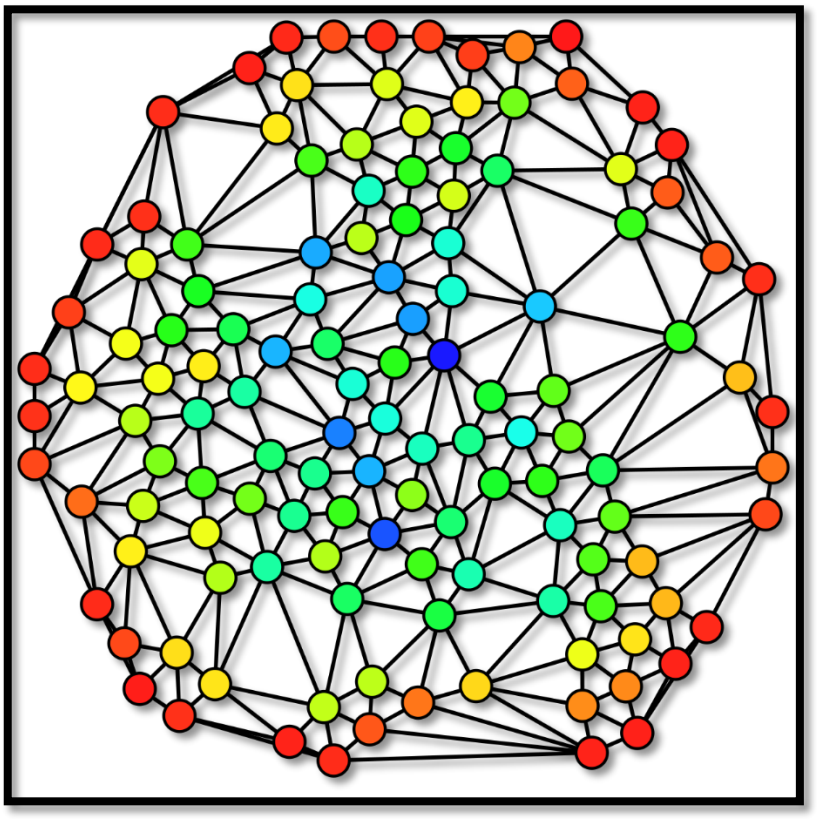


The Girvan-Newman algorithm would remove the edge between nodes **C** and D because it is the one with the highest strength. As you can see intuitively, this means that the edge is located between communities. After removing an edge, the betweenness centrality has to be recalculated for every remaining edge. In this example, we have come to the point where every edge has the same betweenness centrality.

Betweenness centrality

Betweenness centrality measures the extent to which a vertex or edge lies on paths between vertices. Vertices and edges with high betweenness may have considerable influence within a network by virtue of their control over information passing between others.

The calculation of betweenness centrality is not standardized and there are many ways to solve it. It is defined as the number of shortest paths in the graph that pass through the node or edge divided by the total number of shortest paths.



The image above shows an undirected graph coloured based on the betweenness centrality of each vertex from least (red) to greatest (blue).

Louvain algorithm

The Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.

The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

Louvain Community Detection Algorithm is a simple method to extract the community structure of a network. This is a heuristic method based on modularity optimization.

The algorithm works in 2 steps. On the first step it assigns every node to be in its own community and then for each node it tries to find the maximum positive modularity gain by moving each node to all of its neighbour communities. If no positive gain is achieved the node remains in its original community.

The modularity gain obtained by moving an isolated node into a community can easily be calculated by the following formula:

$$\Delta Q = \frac{k_{i,in}}{2m} - \gamma \frac{\Sigma_{tot} \cdot k_i}{2m^2}$$

where m is the size of the graph, $k_{i,im}$ is the sum of the weights of the links from i to nodes in c, k_i is the sum of the weights of the links incident to node i, Σ_{tot} is the sum of the weights of the links incident to nodes in c is the resolution parameter

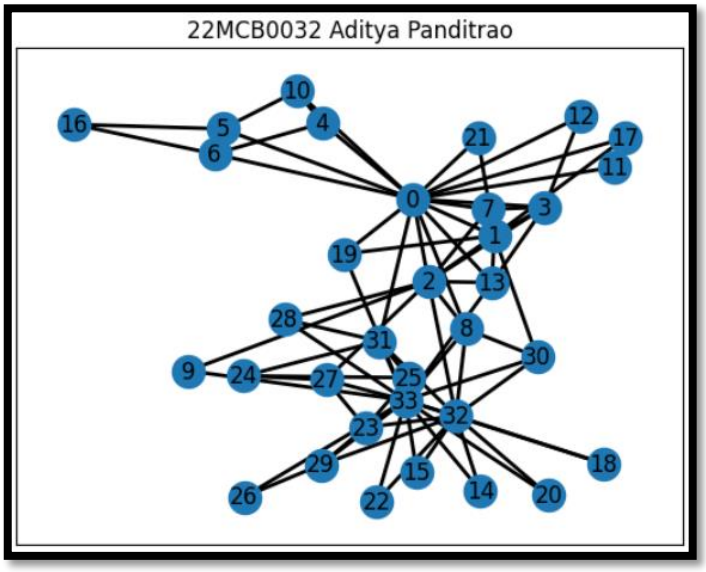
CODING

Step- 1 Importing the necessary libraries

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
from heapq import heappush, heappop
from itertools import count
from networkx.algorithms.community.centrality import girvan_newman
```

Loading the network:

```
[15] # Original Zachary's Karate Club graph
     G = nx.karate_club_graph()
     pos = nx.spring_layout(G, k=0.1, iterations=30, scale=1.3)
     nx.draw_networkx_nodes(G, pos=pos)
     nx.draw_networkx_labels(G, pos=pos)
     nx.draw_networkx_edges(G, pos=pos, width=2,alpha=1,edge_color='k')
     plt.title("22MCB0032 Aditya Panditrao")
     plt.show()
```

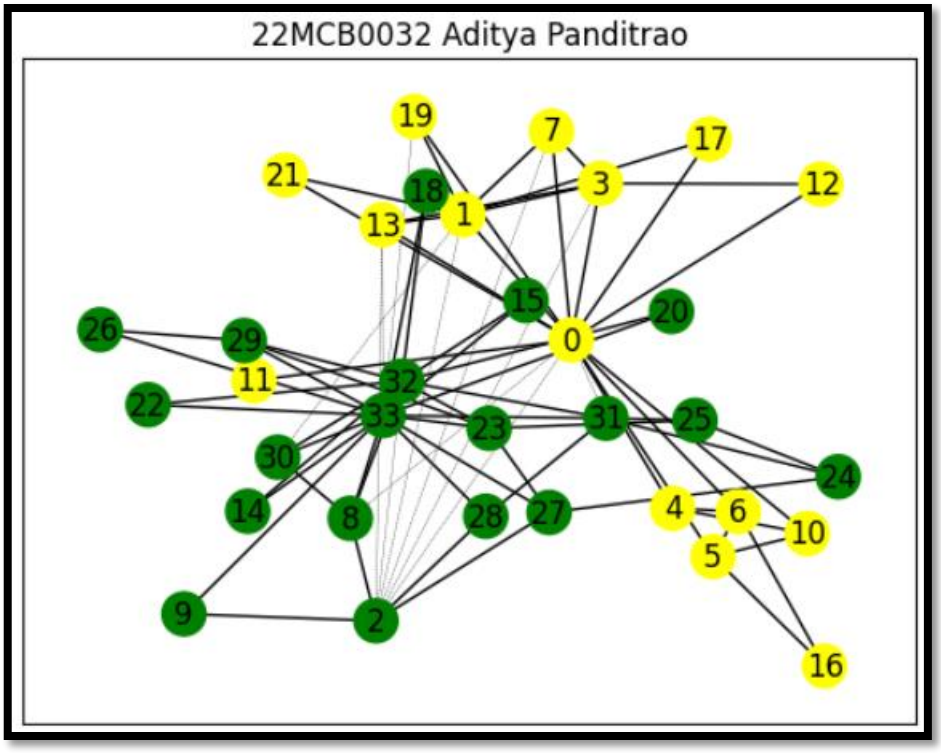


Girvan Newman Algorithm Implementation:

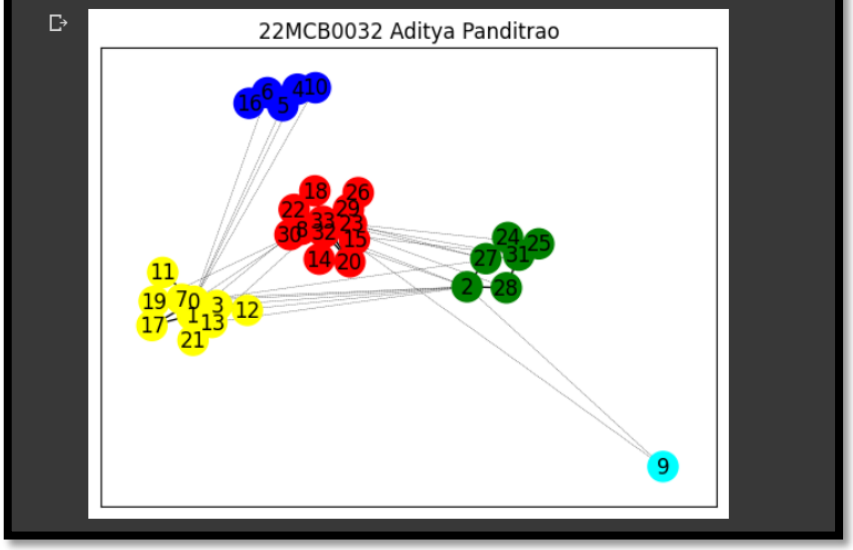
Girvan Newman Algorithim

```
def visualize_community_structure(num_communities, G):
  communities, removed_edges = girvan_newman(G,num_communities)
  G = get_remaining_edges(G, removed_edges)
  node_groups = []

  colors = ['yellow', 'green','blue','red','cyan']

  for i in communities:
    node_groups.append(list(i))

  color_map = []
  for node in G:
    for i in range(len(node_groups)):
      if node in node_groups[i]:
        color_map.append(colors[i])

  pos = nx.spring_layout(G, k=0.1, iterations=15, scale=10)
  nx.draw_networkx_nodes(G, pos=pos, node_color=color_map)
  nx.draw_networkx_labels(G, pos=pos)
  nx.draw_networkx_edges(G, pos=pos, edgelist=removed_edges, width=0.25, edge_color='k', style='dashed')
  nx.draw_networkx_edges(G, pos=pos, width=1,alpha=1,edge_color='k')
  plt.title("22MCB0032 Aditya Panditrao")
  plt.show()
```

```
[17] # Zachary's Karate Club graph with 2 communities
     visualize_community_structure(num_communities=2, G = nx.karate_club_graph())
```

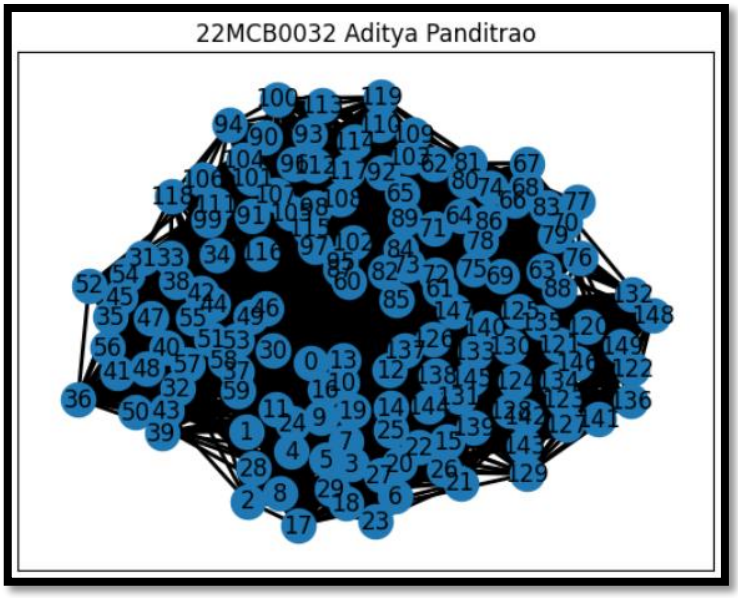


```
# Zachary's Karate Club graph with 5 communities
visualize_community_structure(num_communities=5, G = nx.karate_club_graph())
```

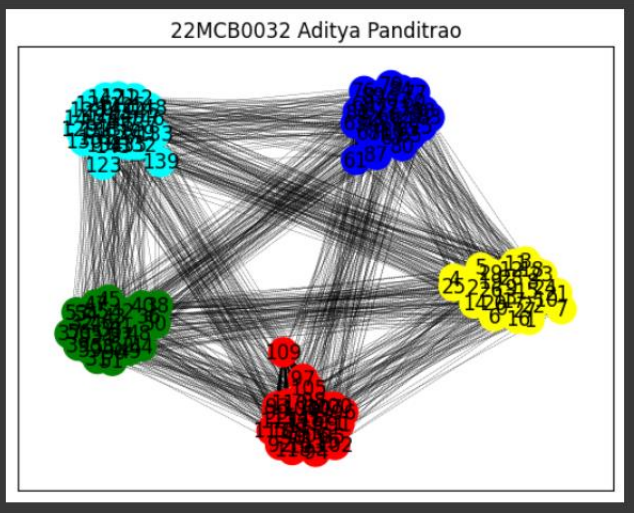


Planted L-Partition

```
[21] # Original Planted L-Partition Graph
     G = nx.planted_partition_graph(5,30,0.8,0.1)
     pos = nx.spring_layout(G, k=0.1, iterations=30, scale=1.3)
     nx.draw_networkx_nodes(G, pos=pos)
     nx.draw_networkx_labels(G, pos=pos)
     nx.draw_networkx_edges(G, pos=pos, width=2,alpha=1,edge_color='k')
     plt.title("22MCB0032 Aditya Panditrao")
     plt.show()
```



```
# Planted L-Partition Graph graph with 5 communities
visualize_community_structure(num_communities=5, G = nx.planted_partition_graph(5,30,0.8,0.1))
```



Spectral Clustering Implementation:

```
SPECTRAL CLUSTERING

[40] import numpy as np
     import scipy.sparse as sp
     import networkx as nx
     import matplotlib.pyplot as plt
     from sklearn.cluster import KMeans

[41] def normalize_eigenvectors(e):
         return e/np.sqrt(np.sum(e**2))
```

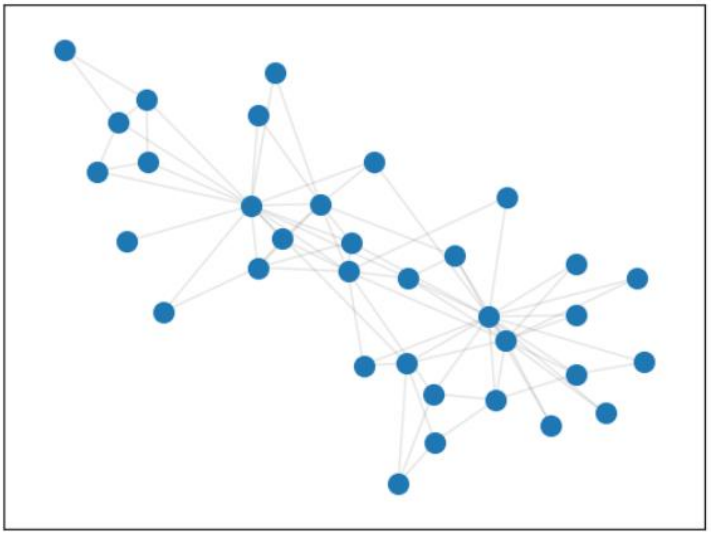
```
def laplacian(G, laplacian_type="unnormalized"):

    D = np.diag(np.sum(np.array(nx.adjacency_matrix(G).todense()),
axis=1))
    W = nx.adjacency_matrix(G).toarray()

    assert D.shape == W.shape, "Shapes of D and W don't match."

    L = None

    if laplacian_type == "unnormalized":
        L = D - W
    elif laplacian_type == "symmetric":
        I = np.ones(D.shape)
        D_inv_root = np.linalg.inv(np.sqrt(D))

        L = I - np.dot(D_inv_root, W).dot(D_inv_root)
    elif laplacian_type == "random_walk":
        I = np.ones(D.shape)
        D_inv = np.linalg.matrix_power(D, -1
```

```
        L = I - np.matmul(D_inv, W)
    else:
        raise ValueError("Laplacian type can be 'normalized' or
'unnormalized'.")

    return L
def generate_labels_dict(G, kmeans):
    num_nodes = len(G.nodes)
    num_clusters = kmeans.n_clusters

    labels_dict = {c: [] for c in range(num_clusters)}

    for i in range(num_nodes):
        labels_dict[kmeans.labels_[i]].append(i)

    return labels_dict
def visualize_graph(G, pos, labels_dict=None, colors=None,
node_size=100, edge_alpha=0.1, labels=False):
    if labels_dict is not None and colors is not None:
        for k, v in labels_dict.items():
            # nodes
            nx.draw_networkx_nodes(G,
                pos,
                nodelist=v,
                node_color=colors[k],
                node_size=node_size
            )
    else:
        nx.draw_networkx_nodes(G, pos, node_size=node_size)

    if labels:
        nx.draw_networkx_labels(G, pos)

    nx.draw_networkx_edges(G, pos, width=1.0, alpha=edge_alpha)
def spectral_clustering(G, k, pos, colors, visualize=True,
laplacian_type="unnormalized", **kwargs):
    # Calculate Laplacian
    L = laplacian(G, laplacian_type=laplacian_type)

    # Get eigenvectors from the Laplacian
    _, eig_vectors = sp.linalg.eigs(L, k)

    # Use eigenvectors as features
    U = eig_vectors.real

    if laplacian_type == "symmetric":
        U = np.apply_along_axis(normalize_eigenvectors, 0, U)
```

```
    # Cluster using KMeans
    kmeans = KMeans(n_clusters=k, random_state=0).fit(U)



    # Get labels
    labels_dict = generate_labels_dict(G, kmeans)

    # Visualize
    if visualize:
        node_size = kwargs.get('node_size', 100)
        edge_alpha = kwargs.get('edge_alpha', 0.1)
        labels = kwargs.get('labels', False)

        visualize_graph(G, pos, labels_dict, COLORS,
node_size=node_size, edge_alpha=edge_alpha, labels=labels)

    return labels_dict
```

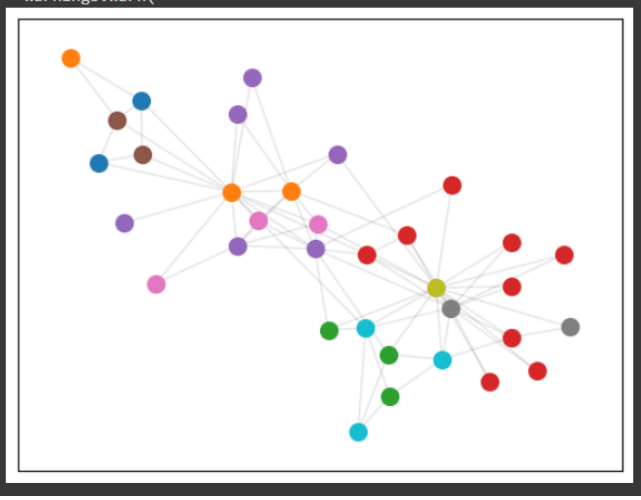
```
[46] COLORS = \
         ["tab:blue", "tab:orange", "tab:green",
          "tab:red", "tab:purple", "tab:brown",
          "tab:pink", "tab:gray", "tab:olive",
          "tab:cyan"]
```

```
G_kk = nx.karate_club_graph()
pos_kk = nx.spring_layout(G_kk)

visualize_graph(G_kk, pos_kk)
```



```
labels_dict = spectral_clustering(G_kk, 10, pos_kk, COLORS, laplacian_type="symmetric")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The c
  warnings.warn(
```



Louvain's Algorithm Implementation

Load the Dataset:

```
DBLP = pd.read_csv('com-dblp.ungraph.txt',
                   sep='\t', header=None, skiprows = 4,
                   names=['node1', 'node2'])
# create graph from edge list
DBLP_network = nx.Graph(DBLP.values.tolist())
def get_graph_info(graph):
    print("Number of nodes:", graph.number_of_nodes())
    print("Number of edges:", graph.number_of_edges())
    print("Average Cluster Coefficients:", nx.average_clustering(graph))
    print("Connected components:",
len(list(nx.connected_components(graph))))
```

```
[102] get_graph_info(DBLP_network)
```

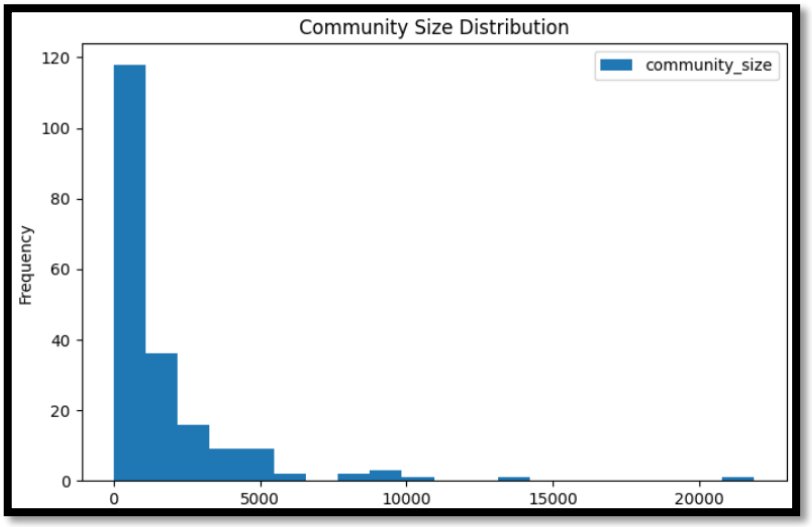
```
Number of nodes: 317080
Number of edges: 1049866
Average Cluster Coefficients: 0.6324308280637396
Connected components: 1
```

```
import community.community_louvain as cl
louvain_communities = cl.best_partition(DBLP_network, random_state=0)
```

```
def convert_communities_dict_to_list(communities_dict):
    unique_partition_count = len(list(set(communities_dict.values())))
    print("Number of unique communities:", unique_partition_count)
    communities = [[] for i in range(unique_partition_count)]
    for node in communities_dict.keys():
        communities[communities_dict[node]].append(node)
    return communities
louvain_communities =
convert_communities_dict_to_list(louvain_communities)
```

```
Number of unique communities: 198
```

```
community_sizes = pd.DataFrame([len(community) for community in
louvain_communities],
                               columns=["community_size"])
community_sizes.plot.hist(bins=20, figsize=(8,5), title="Community Size
Distribution");
```




```
# take a look at the modularity of the Louvain algorithm output
print("Modularity:",
      round(nx_comm.modularity(DBLP_network, louvain_communities), 6))
```

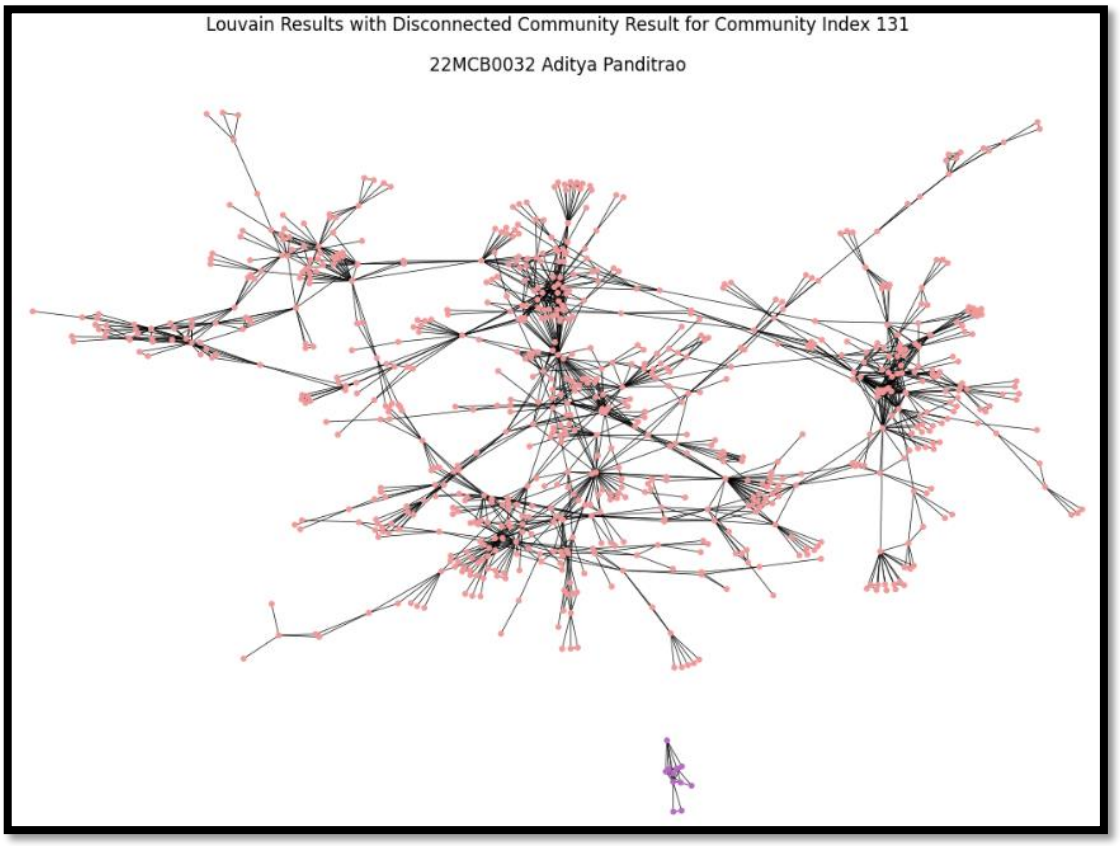
```
Modularity: 0.821751
```

```
# function to check whether the determined communities are connected
def check_community_disconnection(graph, communities):
    total_connected = 0
    disconnected_community_indexes = []
    for i in range(len(communities)):
        if nx.is_connected(graph.subgraph(communities[i])):
            total_connected += 1
```

```
        else:
            disconnected_community_indexes.append(i)
    print("Total Communities:", len(communities))
    print("Total Communities Connected:", total_connected)
    print("Disconnected Community Indexes:",
disconnected_community_indexes)
check_community_disconnection(DBLP_network, louvain_communities)
```

```
Total Communities: 198
Total Communities Connected: 193
Disconnected Community Indexes: [5, 8, 18, 111, 131]
```

```
def create_community_node_colors(graph, communities):
    number_of_colors = len(communities[0])
    colors = ["#EF9A9A", "#BA68C8", "#64B5F6", "#81C784",
              "#FFF176", "#BDBDBD"][:number_of_colors]
    node_colors = []

    for node in graph:
        current_community_index = 0
        for community in communities:
            if node in community:
                node_colors.append(colors[current_community_index])
                break
            current_community_index += 1
    return node_colors
c# get subgraph of community with index 131 for demo
disconnected_subgraph = DBLP_network.subgraph(louvain_communities[131])
disconnected_components =
list(nx.connected_components(disconnected_subgraph))

# create visualization
node_colors = create_community_node_colors(disconnected_subgraph,
disconnected_components)
pos = nx.spring_layout(disconnected_subgraph, iterations=50, k=0.05,
                       seed=2)

plt.figure(1,figsize=(12,8))
nx.draw(disconnected_subgraph,
        pos = pos,
        node_size=10,
        width=0.5,
        node_color=node_colors)
plt.title("Louvain Results with Disconnected Community Result for
Community Index 131")
plt.suptitle("22MCB0032 Aditya Panditrao")
plt.show()
```



Conclusion-

In conclusion, community detection techniques on social networks have become an important research area due to the increasing popularity of online social platforms. These techniques aim to identify cohesive groups of users or nodes within a network based on their connections and interactions.

Various methods have been proposed for community detection, including clustering algorithms, modularity-based approaches, spectral analysis, and more recently, deep learning-based methods. Each technique has its strengths and weaknesses, and the choice of method depends on the characteristics of the network and the research question being addressed.

Community detection has numerous applications, including understanding social dynamics, identifying influential users or groups, detecting anomalous behavior, and predicting user behavior. However, it also raises ethical concerns regarding privacy and bias, as well as challenges related to scalability and the interpretation of results.

Overall, community detection on social networks remains a vibrant and rapidly evolving field, with ongoing efforts to develop more accurate, scalable, and interpretable methods that can better capture the complex and dynamic nature of social interactions.