School of Computer Science and Engineering
(WINTER 2022-2023)

Subject Name: SOCIAL NETWORK ANALYTICS LAB

Assessment No: 2

Community detection algorithm

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

Student Name: **Anuraj Bose**

Reg No: **22MCB0011**

Email: anuraj.bose2022@vitstudent.ac.in
Mobile:9073137517

Algorithms for Community Detection Comparative Analysis

Title: Community Detection Algorithms: A Comprehensive Report

AIM :

To understand the dynamics and adjustment of Community Detection Algorithm

Python NetworkX for Community Detection Algorithms Understanding

Based on the node connection patterns, techniques for community discovery are employed to locate groupings or communities in networks. Popular Python library NetworkX offers several methods for interacting with networks, such as community discovery techniques. Using NetworkX to apply community discovery methods on network data will be covered in this tutorial.

Abstract:

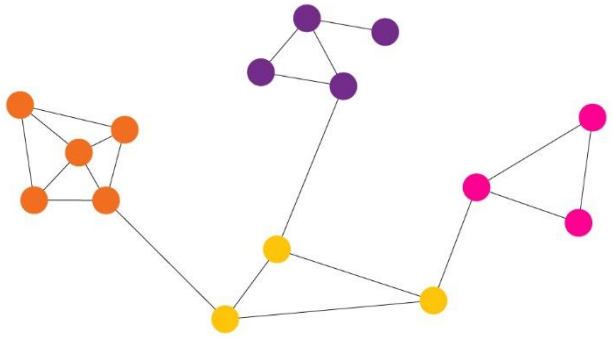
Identifying communities or classes within a network of nodes based on node connection patterns is known as community detection, and it is a crucial job in network analysis. underlying concepts, benefits, and applications of community classification algorithms are covered in detail in this research. We check a few well-known algorithms, such as Girvan-Newman, Louvain, Label Propagation, and Clauset-Newman-Moore, emphasising their salient features and implementation information. We also look at the difficulties and assessment criteria related to community detection. The purpose of the paper is to give readers a thorough grasp of community identification methods and their possible uses in many fields.

INTRODUCTION :

Despite the fact that people are excellent at spotting unique or recurring patterns among a small number of components, the architecture of massive, linked networks makes it nearly difficult to carry out such fundamental tasks by hand. Visually identifying clusters of tightly linked nodes is straightforward, but programming these jobs requires more advanced techniques. In different networks, these clusters of tightly linked components are located using community detection techniques.

The Girvan-Newman method, developed by M. Girvan and M. E. J. Newman, is one of the most extensively used community discovery techniques. They claim that node groups in a network are closely connected to one another within communities and distantly connected between communities.

The fundamental ideas behind community detection methods, their particular implementations.



Methods of Community Detection

Agglomerative and divisive community detection techniques are the two primary categories.

Typically, **aggregative approaches** begin with a network that only has nodes from the original graph. Stronger edges are given precedence over weaker ones as the edges are added one at a time to the graph. Depending on how an algorithm is implemented, the weight or strength of an edge is determined differently.

Conversely, **divisive approache**s rely on the method of repeatedly eliminating edges from the initial graph. Edges are eliminated in ascending order of strength. Since the weight of the remaining edges varies once an edge is removed, the edge-weight computation must be done again at each step. Communities, or groupings of tightly linked nodes, are what result after a certain number of steps.

NetworkX's Community Detection

Girvan-Newman algorithm: By gradually eliminating edges from the initial network, the Girvan-Newman method finds communities.

The fluid communities method is based on the fundamental notion of fluids expanding and pushing against one another as they interact in a given environment.

An approach for ***semi-supervised machine learning*** called label propagation gives labels to previously unlabeled data points.

Clique Percolation Algorithm: This algorithm uses the percolation technique to locate k-clique communities in a graph.

The Kernighan-Lin method divides a network into two sets while minimizing the edge cut between the two sets by repeatedly exchanging pairs of nodes.

Girvan-Newman Algorithm

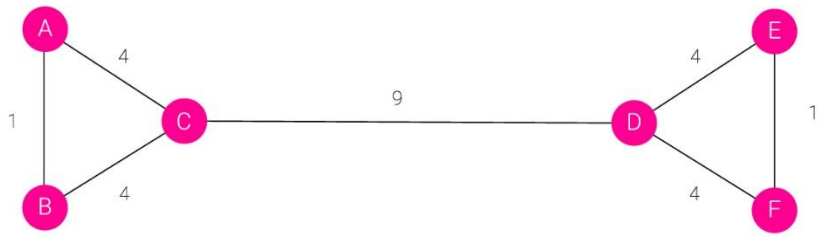
The Girvan-Newman technique depends on the repeated removal of edges with the maximum number of shortest pathways between nodes passing through them for the discovery and analysis of community structures. As edges are eliminated one at a time from the graph, the network fragments into smaller units known as communities. Mark Newman and Michelle Girvan presented the algorithm.

Working Procedure:

By identifying edge betweenness centrailities, the goal is to determine which edges in a network occur more frequently between other pairs of nodes. Therefore, a considerable edge betweenness is anticipated on the edges linking communities. Once the edges with the highest betweenness are removed, the network's underlying community structure will be considerably more fine-grained, making communities much easier to identify.

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

1. For every edge in a graph, calculate the edge betweenness centrality.
2. Remove the edge with the highest betweenness centrality.
3. Calculate the betweenness centrality for every remaining edge.
4. 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/weight 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.

Interdependence centrality/Between Centrality:

The degree to which a node or edge resides on pathways between nodes is measured by betweenness centrality. High betweenness nodes and edges may have significant control on the information moving between other nodes and edges in a network.

There are several approaches to solving the betweenness centrality calculation because it is not standardised. It is determined by dividing the total number of shortest pathways in the graph by the number of shortest paths that go through a node or edge.

Examples of Community Detection Applications:

Community identification algorithms have made their way into a variety of sectors, ranging from social network research to public health campaigns, because networks constitute a crucial component of many real-world situations.

Tracking the contacts and activities of terrorist groups on social media is a well-known use case. Similar to this, dangerous bot groups may be found on social media websites.
Studying the dynamics of particular populations that are vulnerable to epidemic illnesses can be done through community detection. To identify commonalities among patients, comparable research may be done on other types of disorders.
In one of the most recent application cases, community evolution prediction, changes in network architecture are anticipated.

Pseudocode

In each iteration, calculate the betweenness centrality for all edges in the graph. Remove the edge with the highest centrality. Repeat until there are no more edges left.

Calculate the betweenness centrality for each iteration for each edge in the graph. Remove the edge that is most centrally located. Continue until there are no longer any edges.

```
REPEAT
LET n BE number of edges in the graph
FOR i=0 to n-1
LET B[i] BE betweenness centrality of edge i
IF B[i] > max_B THEN
max_B = B[i]
max_B_edge = i
ENDIF
ENDFOR
REMOVE edge i FROM graph
UNTIL number of edges in graph is 0
```

Community Detection Algorithms in NetworkX

girvan_newman(G, most_valuable_edge=None)

1.Newman-Girvan algorithm

A popular approach for community discovery in networks is the Newman-Girvan algorithm, sometimes referred to as the Girvan-Newman algorithm or the edge betweenness algorithm. In 2002, Mark Newman and Michelle Girvan created it. The basis for the method is the idea of "edge betweenness centrality," which quantifies the significance of edges in tying together various groups. The fundamental tenet is that edges with high betweenness centrality are more likely to be located between communities and may thus be thought of as prospective community borders. The steps of the Newman-Girvan algorithm are as follows: Determine the network's betweenness centrality for each edge. This entails figuring out how many of the shortest routes between every pair of nodes cross via each edge. Take out of the network the edge with the highest betweenness centrality. In essence, this splits the network into two or more halves. Calculate the betweenness centrality for each of the revised network's remaining edges. Repeat steps 2 and 3 as necessary to eliminate all edges or to locate the required number of communities.

Communities or subnetworks are the resulting separated components.

NAME : ANURAJ BOSE | REGISTRATION NUMBER : 22MCB0011 | SUBJECT : SOCIAL NETWORK ANALYTICS | TOPIC : COMMUNITY DETECTION ALGORITHM IN SOCIAL NETWORK GROUPS

```
import os
import networkx as nx
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import matplotlib.pyplot as plt

# Step 1: Load the text dataset
# Assuming you have a .txt file named 'dataset.txt' containing one document per line
with open('/content/group.csv', 'r') as file:
    documents = file.readlines()
```

Step 2: Preprocess the text data

```
[4]
# Assuming you have already preprocessed the text data by removing stopwords, punctuation, etc.
# If not, you can implement preprocessing steps here
```

Step 3: Create a document-term matrix

```
[5]  vectorizer = CountVectorizer()
     X = vectorizer.fit_transform(documents)
```

Step 4: Apply LDA for topic modeling

```
[6]  lda = LatentDirichletAllocation(n_components=5)  # Assuming 5 topics
     lda.fit(X)
```

```
LatentDirichletAllocation(n_components=5)
```

Step 5: Extract topic distributions for documents

```
[7]  topic_dist = lda.transform(X)
     topic_labels = topic_dist.argmax(axis=1)
```

Step 6: Create a graph representation of the documents

```
[8]  G = nx.Graph()
     for i, document in enumerate(documents):
         G.add_node(i, text=document, topic=topic_labels[i])
```

Step 7: Apply the Girvan-Newman algorithm for community detection

```
[9]  communities = nx.community.girvan_newman(G)
```

Step 8: Get the final community partition

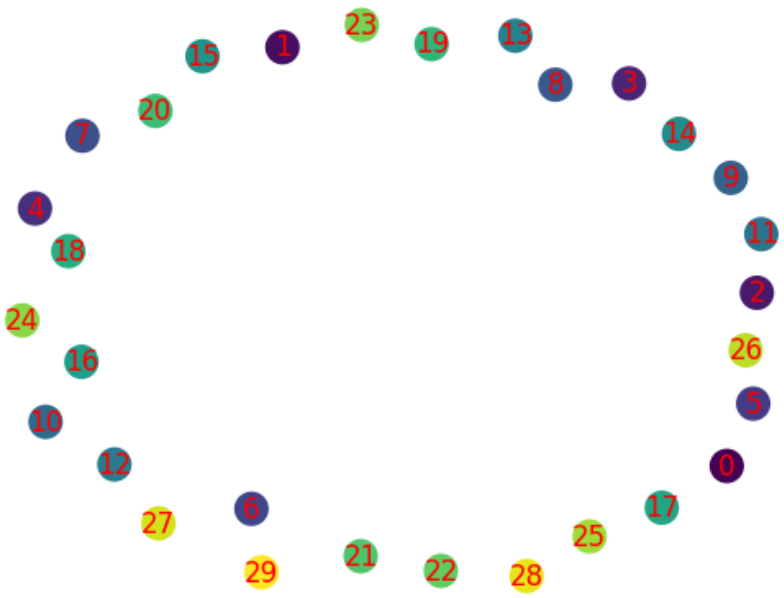
```
[10] partition = next(communities)
```

Step 9: Visualize the graph with community colors

```
[11] pos = nx.spring_layout(G)
```

```
[14] # Draw nodes with different community colors
     node_colors = [idx for idx, comm in enumerate(partition) for _ in comm]
     nx.draw_networkx_nodes(G, pos, node_color=node_colors, cmap='viridis', node_size=200)

     # Draw edges
     nx.draw_networkx_edges(G, pos, alpha=1)

     # Draw node labels
     nx.draw_networkx_labels(G, pos, font_color='red')

     # Show the plot
     plt.axis('off')
     plt.show()
```

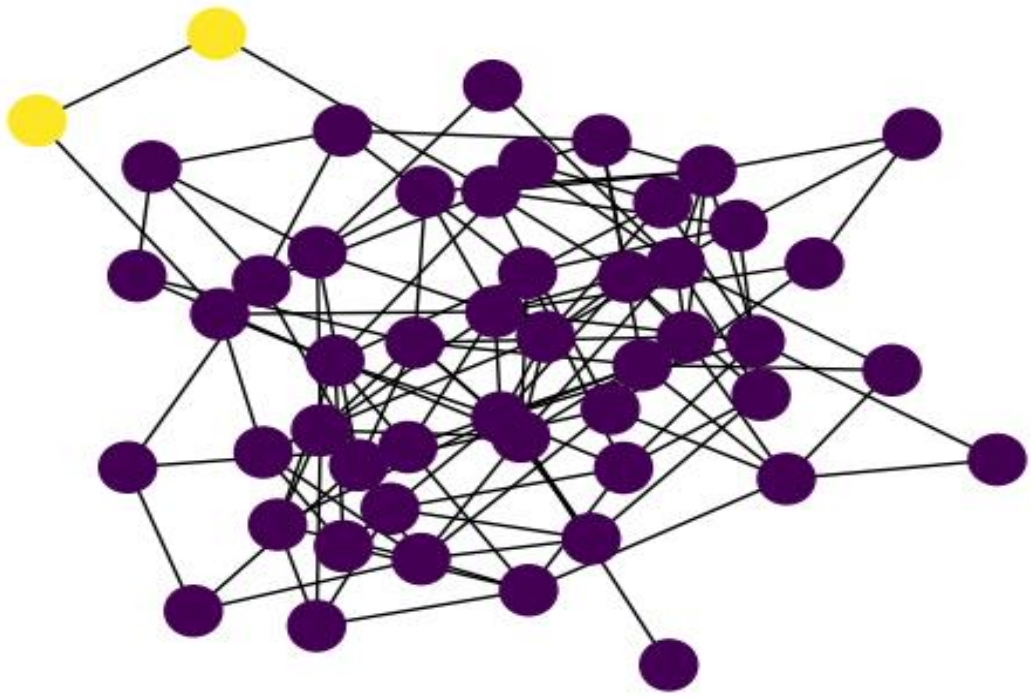
OUTPUT:



1.Program code:

[16]
```python
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms import community

# Generate a random graph
G = nx.erdos_renyi_graph(50, 0.1)

# Apply the Newman-Girvan algorithm
comp = community.girvan_newman(G)

# Select the top-level community
top_level_communities = next(comp)

# Convert the top-level community into a dictionary
partition = {}
for idx, community_nodes in enumerate(top_level_communities):
    for node in community_nodes:
        partition[node] = idx

# Create a layout for visualizing the graph
layout = nx.spring_layout(G)

# Draw the nodes with community colors
node_colors = [partition[node] for node in G.nodes()]
nx.draw_networkx_nodes(G, layout, node_color=node_colors, cmap='viridis')

# Draw the edges
nx.draw_networkx_edges(G, layout)

# Display the graph
plt.axis("off")
```

OUTPUT:



2. **Louvain algorithm**

The Louvain method is a well-liked community recognition technique that seeks to identify a network's modular structure. By repeatedly merging and reassigning nodes to communities, this iterative approach maximises the network's modularity. The Louvain algorithm's general stages are as follows:

Initialization: Assign each network node to a distinct community at the beginning.

Determine the network's current state's modularity by doing the aforementioned computation. Modularity evaluates how well a network is divided into communities. It contrasts the quantity of edges inside communities with the quantity of edges anticipated in a random network.

Iteration:

Calculate the increase in modularity for each node when it is moved from its current community to one of its neighbouring communities. The increase in

modularity that would follow from shifting the node is known as the modularity gain.
Transfer the node to the community where modularity gains are the greatest. Leave the node in its present community if the modularity benefit is negative or zero for all surrounding communities.

Repeat the previous steps for each node in the network.
Community aggregation: After performing a single pass over all nodes, each community is regarded as a single node in a new network. The weights of the edges between communities are the weights of the initial edges between the nodes in the communities.

Iterate the stages of node reassignment, community aggregation, and modularity computation until no further increase in modularity is seen. This indicates that the algorithm has identified the ideal community structure and has converged.

Post-processing: The final community structure is generated once the Louvain method has converged. Nodes that are part of the same community are combined.

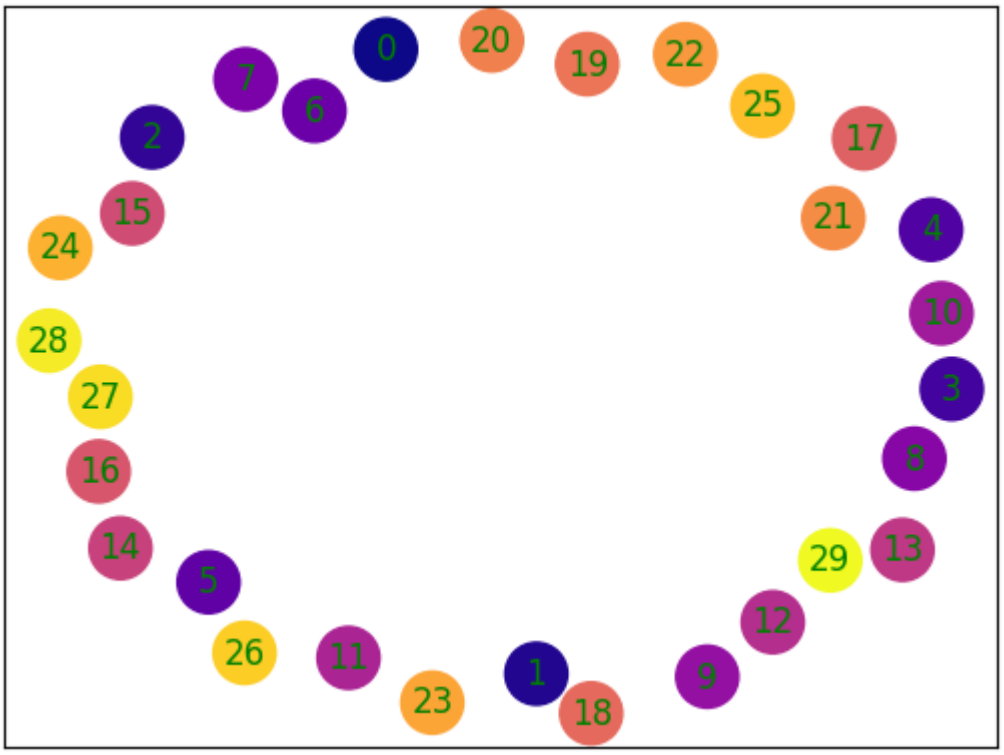
It's worth noting that the Louvain algorithm is a heuristic method and may not always find the global optimum, but it often produces good results in practice.

Program Code:

```
[19] import networkx as nx
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.decomposition import LatentDirichletAllocation
     from community import community_louvain
     import matplotlib.pyplot as plt

     # Step 1: Load the text dataset
     with open('/content/group.csv', 'r') as file:
         documents = file.readlines()

     # Step 2: Preprocess the text data

     # Step 3: Create a document-term matrix
     vectorizer = CountVectorizer()
     X = vectorizer.fit_transform(documents)

     # Step 4: Apply LDA for topic modeling
     lda = LatentDirichletAllocation(n_components=5)  # Assuming 5 topics
     lda.fit(X)

     # Step 5: Extract topic distributions for documents
     topic_dist = lda.transform(X)
     topic_labels = topic_dist.argmax(axis=1)

     # Step 6: Create a graph representation of the documents
     G = nx.Graph()
     for i, document in enumerate(documents):
         G.add_node(i, text=document, topic=topic_labels[i])
```

```
# Step 7: Apply the Louvain algorithm for community detection
partition = community_louvain.best_partition(G)
# Step 8: Visualize the graph with community colors
pos = nx.spring_layout(G)

# Get unique community labels
community_labels = set(partition.values())

# Draw nodes with different community colors
node_colors = [partition[node] for node in G.nodes()]
nx.draw_networkx_nodes(G, pos, node_color=node_colors, cmap='plasma', node_size=500)

# Draw edges with community colors
edge_colors = ['blue' if partition[edge[0]] != partition[edge[1]] else 'viridis' for edge in G.edges()]
nx.draw_networkx_edges(G, pos, alpha=0.5, edge_color=edge_colors)

# Draw node labels
nx.draw_networkx_labels(G, pos, font_color='green')

# Show the plot
plt.axis('on')
plt.show()
```

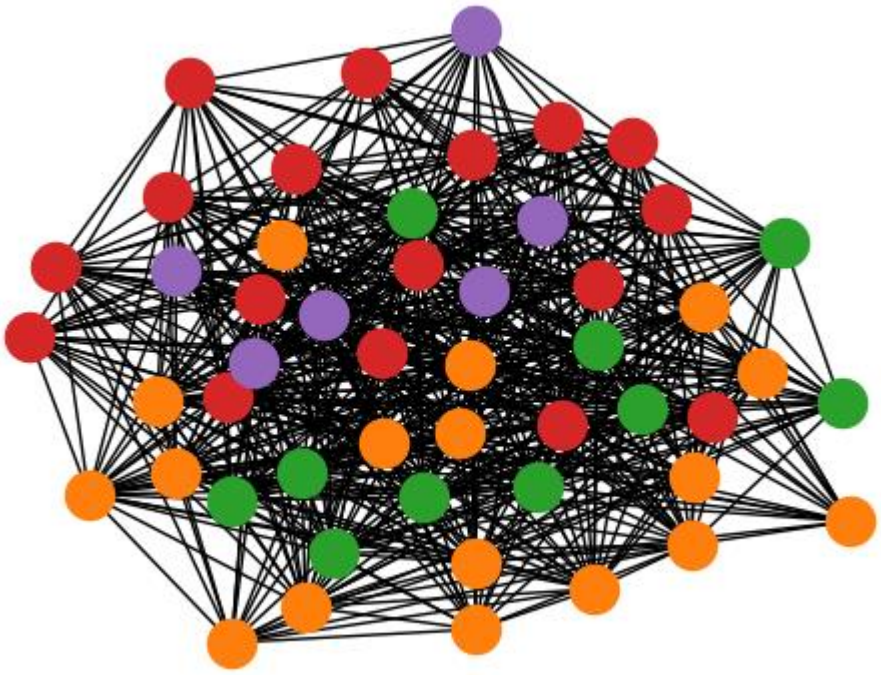
OUTPUT:



Program code:

```
[23] from community import community_louvain
     import matplotlib.cm as cm
     import matplotlib.pyplot as plt
     import networkx as nx

     # Generate a random graph
     G = nx.erdos_renyi_graph(50,0.5)

     # Apply the Louvain community detection algorithm
     partition = community_louvain.best_partition(G)

     # Create a layout for visualizing the graph
     layout = nx.spring_layout(G)

     # Draw the nodes with community colors
     for node, community_id in partition.items():
         nx.draw_networkx_nodes(G, layout, [node], node_color=f"C{community_id+1}")

     # Draw the edges
     nx.draw_networkx_edges(G, layout)

     # Display the graph
     plt.axis("off")
     plt.show()
```

OUTPUT:



Conclusion :

For graph analysis, community discovery is a useful technique. These algorithms have been incorporated into several real-world use cases, including healthcare programmes and the identification of terrorists.

When it comes to analysing graph networks and performing sophisticated algorithms like community discovery, the Python NetworkX module offers extensive functionality. However, you should take into account employing a graph database in conjunction with NetworkX if you're trying to operationalize your graph algorithms and are searching for functionality like incremental updates, data persistency, and higher speed.