Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Social Network Analysis Laboratory (DJ19DSL8014)

AY: 2024-25

Bhuvi Ghosh
60009210191

Experiment 7

Aim: Implementation clique percolation algorithm for community detection.

Theory:

1. Introduction

In the realm of network analysis, the discovery of community structures within large networks is a significant challenge with profound implications across various fields such as sociology, biology, and computer science. The Clique Percolation Method (CPM) stands out as a sophisticated approach for detecting overlapping communities within complex networks. This method, premised on the idea that social groups overlap and share common ties, treats communities as groups of nodes that form cliques (fully connected subgraphs). By iteratively identifying and merging these cliques based on shared nodes, CPM reveals the intricate tapestry of communities that would otherwise be obscured in the complexity of network data. The following sections will delve into the mechanics of CPM, providing the theoretical foundations and algorithmic steps essential for implementing this powerful method in practice.

Input:

- A network graph G(V, E), where V is the set of vertices and E is the set of edges.
- An integer k representing the size of the clique for percolation.

Output:

- A set of communities C, where each community is a subset of V.

1. Find All k-Cliques: Identify all possible k-cliques in the graph. A k-clique is a complete subgraph of k vertices.

2. Create Clique Graph: Construct a clique graph G'(V', E') where each k-clique is represented as a vertex in V'. Two vertices in G' are connected if their corresponding k-cliques in G share k-1 vertices.

3. Identify k-Clique Communities: Find connected components in G'. Each connected component represents a k-clique community. This step is performed using a breadth-first search (BFS) or depth-first search (DFS) algorithm on G'.

4. Merge k-Clique Communities: For each pair of connected components (communities) in G', merge them into a single community if they share k-1 vertices.

5. Resulting Communities: The resulting connected components after the merger represent the communities within the original graph G.

6. Stopping Criterion: Repeat step 4 until no further merging is possible, i.e., when each pair of communities shares less than k-1 vertices.

Lab Assignment

For above network assume k=3 and detect communities using CPM.

https://medium.com/@sabadejuyee21/community-detection-through-clique-percolation-method-on-the-game-of-thrones-universe-b12efb188d99

```python
import networkx as nx
import matplotlib.pyplot as plt
from itertools import combinations

# Step 1: Define the graph
G = nx.Graph()
edges = [
    (1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (4, 6), (5, 6),
    (5, 7), (5, 9), (6, 8), (7, 8), (7, 9), (7, 10)
]
G.add_edges_from(edges)

# Step 2: Find all k-cliques
k = 3
cliques = [clique for clique in nx.find_cliques(G) if len(clique) >= k]

# Extract all k-sized cliques
k_cliques = []
for clique in cliques:
    k_cliques += list(combinations(clique, k))

# Step 3: Build the clique graph
clique_graph = nx.Graph()
for idx, clique in enumerate(k_cliques):
    clique_graph.add_node(idx, members=set(clique))

# Add edges between cliques that share k-1 nodes
for i in range(len(k_cliques)):
    for j in range(i + 1, len(k_cliques)):
        if len(set(k_cliques[i]) & set(k_cliques[j])) >= k - 1:
            clique_graph.add_edge(i, j)
```

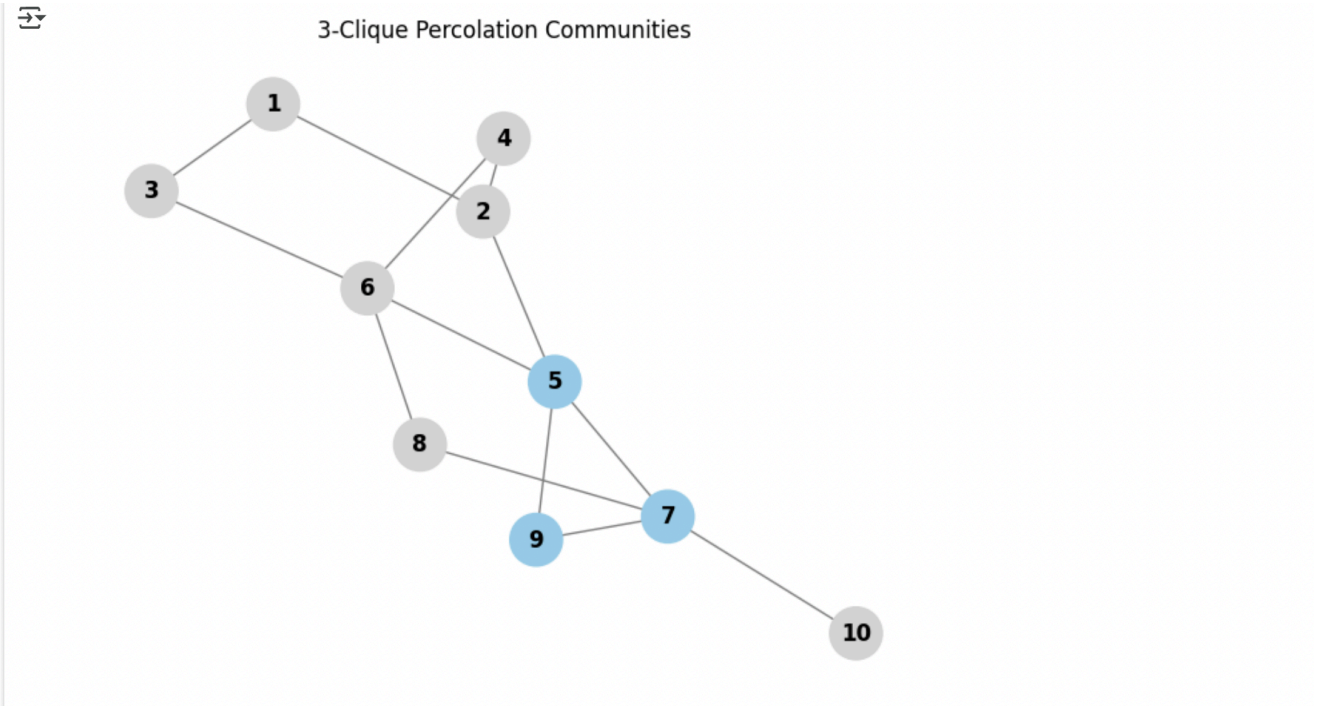
```python
# Step 4: Find connected components in clique graph (each is a community)
communities = []
for component in nx.connected_components(clique_graph):
    community_nodes = set()
    for node in component:
        community_nodes.update(clique_graph.nodes[node]['members'])
    communities.append(community_nodes)

# Step 5: Visualize the communities
pos = nx.spring_layout(G, seed=42)
colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'violet', 'gold']
node_color_map = {}

for i, comm in enumerate(communities):
    for node in comm:
        node_color_map[node] = colors[i % len(colors)]

# Assign gray to nodes not in any community
for node in G.nodes():
    if node not in node_color_map:
        node_color_map[node] = 'lightgray'

# Draw the graph
node_colors = [node_color_map[n] for n in G.nodes()]
nx.draw(G, pos, with_labels=True, node_color=node_colors, node_size=800, font_weight='bold', edge_color='gray')
plt.title(f"{k}-Clique Percolation Communities")
plt.show()
```



Conclusion: The Clique Percolation Method (CPM) successfully identified overlapping communities in the network by detecting and connecting 3-cliques (fully connected groups of 3 nodes). By linking cliques that shared 2 nodes, we formed larger, meaningful community structures. This method effectively captures real-world scenarios where nodes (e.g., people) can belong to multiple groups, making CPM a powerful tool for community detection in complex networks.