Community Detection in Social Networks using Ant Colony Algorithm and Fuzzy Clustering

Shri Harie Vignesh R D - CS23B2033
Amit Anil Kamble - CS23B2034

Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

November 2024

Table of Contents

Problem Definition - Community Detection

- **Goal:** Detect Communities or densely connected groups of nodes in social networks.
- A common meaning of a community is that it consists of a subset of nodes from the original network such that the number of edges between nodes in the same community is large and the number of edges connecting nodes in different communities is small.
- **Problem Representation:** Given a graph $G = (V, E)$, identify clusters $G_1, G_2, \dots, G_k$ such that

$$G_1 \cup G_2 \cup \cdots \cup G_k = G \quad \text{and} \quad G_1 \cap G_2 \cap \cdots \cap G_k = \emptyset \tag{1}$$

where intra-community edges are maximized and inter-community edges are minimized.

Problem Definition - Community Detection

- Community detection is a very computationally intensive task.
- Therefore, in such cases it is prevalent to use heuristic algorithms.
- **Heuristic algorithms:**These algorithms return approximate solution but have an advantage of lower time complexity.
- It makes the analysis of larger networks feasible.
- A popular technique for community detection models the problem as an optimization task, maximizing modularity to assess community structure.
- **Modularity (Q):** Measure used for the quality of detected communities.

Objective

- **Primary Objective:** Maximize modularity to enhance community quality.
- **Improvement Goal:** Use ACO for initial community detection and FCM for refining results, achieving a solution competitive with state-of-the-art methods.

Basic Optimization Method - Ant Colony Optimization (ACO)

- **Ant Colony Optimization (ACO)** is bio-inspired algorithm used to solve optimization problems by mimicking the foraging behavior of ants.ACO uses artificial ants that lay pheromones on edges to help find clusters within a graph.
- ACO uses artificial ants that lay pheromones on edges to help find clusters within a graph.
- The algorithm builds on previous pheromone trails to construct communities, refining the solution iteratively.
- The algorithm is divided into 3 main phases.which will discuss in furthur slides.

Basic Optimization Method - Fuzzy Clustering (Fuzzy C-Means)

- **Fuzzy C-Means (FCM)** is used to fine-tune the results by assigning nodes to communities with varying membership degrees.
- **Objective Function:** Minimizes the following:

$$J(U, V) = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^{m} \|x_i - v_j\|^2$$

  where u_{ij} represents the degree of membership, and v_j is the center of the j-th cluster.
- **Process:** FCM re-assigns nodes iteratively until convergence based on the cluster membership values.

Method to Achieve Objective

- The input to the algorithm is undirected graph G = (V,E) and the output is weighted graph ,where each vertex in weighted graph represents a community and the edge set represents the edges between different communities.
- There are 3 main phases namely Exploration, Construction, Optimization.
- **Exploration:** In the exploration phase, ants traverse the graph, leaving pheromone on edges.
- **Construction:** The construction phase then uses pheromone levels to form initial communities after Exploration phase.
- **Optimization:** Now in Optimization phase improves the solution produced by construction phase. Finally, Fuzzy C-Means (FCM) clustering fine-tunes the community structure for enhanced results.

Action Plan - Detailed Steps

- **Step 1: Initialization**
  - Initialize the undirected graph $G = (V, E)$ and assign initial pheromone levels to edges.
  - Define the number of ants and their starting positions.
  - Set initial values for the parameters α, β, and ρ (pheromone importance, heuristic importance, and evaporation rate).

Action Plan - Detailed Steps

- **Step 2: Exploration with Ant Colony Optimization**
  - In this phase, artificial ants move along the edges of the graph, leaving pheromone trails as they go.
  - The purpose of these pheromones is to highlight edges that connect nodes likely to belong to the same community.
  - As ants travel, the edges with high pheromone levels become more attractive, suggesting that these edges link nodes in the same community.
  - Ants are moved in parallel steps, repeating this process until they complete a specific number of steps. At intervals, the pheromone on edges is updated to increase efficiency.
  - An ant selects an edge (i, j) with a probability that depends on:
    - **Pheromone Level** τ_{ij}: This represents the strength of the trail left by previous ants on edge (i, j). A higher pheromone level makes the edge more attractive.
    - **Heuristic Information** η_{ij}: This is based on modularity, a measure of how well the edge contributes to a good community structure.

Action Plan - Detailed Steps

- The probability $p(i,j)$ that an ant will select edge (i,j) is given by:

$$p(i,j) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{q \in N_i} \tau_{iq}^{\alpha}\eta_{iq}^{\beta}}$$

  where:

  - α is the importance of pheromone in deciding the path.
  - β is the importance of heuristic information.
  - N_i is the set of neighbors of node i.
- This formula indicates that each edge's probability is a ratio of its pheromone and heuristic values compared to all other edges from node i.
- The heuristic η_{ij} also depends on modularity gain $\Delta Q(i,j)$, which measures how much better the community structure becomes if nodes i and j are connected.

Action Plan - Detailed Steps

- **Heuristic Information Formula:**

$$\eta_{ij} = \begin{cases} \Delta Q(i,j) + e.phm, & \text{if } \Delta Q(i,j) + e.phm > 0 \\ 0, & \text{otherwise} \end{cases}$$

- where $\Delta Q(i,j)$ is the modularity gain, representing the improvement in community structure if nodes i and j are connected, and $e.phm$ is an additional factor related to pheromone levels.

Action Plan - Detailed Steps

- After each step, the pheromone level on edges traversed by ants is updated. This update includes:
  - **Incrementing pheromone** on edges that ants have traveled more frequently.
  - Adding the connected vertex to a **tabu list**, which prevents the ant from revisiting it immediately.
- At the end of each iteration:
  - **Pheromone Evaporation**: The pheromone strength is reduced by 7% to prevent paths from becoming overly dominant.
  - **Reset Ants**: Ants are repositioned to start fresh for the next iteration.

Action Plan - Detailed Steps

- **Step 3: Construction of Primary Communities**
- **Sort the Edges by Pheromone Level:**
  - The algorithm sorts all edges in descending order by their pheromone levels, so edges with higher pheromone are processed first.
  - This sorting helps ensure that the most promising connections (edges with strong pheromone levels) are used to define the communities early on.
- **Edge Processing:**
  - If neither node has been assigned a community, they form a new community.
  - If one node is already in a community, the other node is added to that community.
  - If both nodes are in separate communities,
    - A weighted connection (edge) is created between these two communities if it doesn't exist.
    - If there is already a connection, it increments the weight of this edge to reflect the increased connection strength.

Action Plan - Detailed Steps

- **Update Community Graph:**
  - During this phase, the weight of each connection between communities is also updated based on the pheromone levels of the edges connecting them.
- **Result:**
  - At the end of this phase, the weighted graph represents the basic community structure, where:
    - **Nodes** represent communities.
    - **Weighted edges** represent pheromone levels, indicating the strength of connections between communities.

Action Plan - Detailed Steps

Step 4:Local Optimization Phase:

- We begin with the initial set of communities created in the previous phase, which were based on pheromone levels.
- However, since this phase didn't fully use the actual graph structure, the resulting communities may not be ideal.

Purpose of Local Optimization:

- This phase examines each node (or "vertex") to see if moving it to a different community would make the communities stronger and more meaningful.

How Nodes are Evaluated:

- Each node has two types of connections:
  - **Internal degree**: Links to other nodes within the same community.
  - **External degree**: Links to nodes in different communities.
- The algorithm calculates the **total external degree** for each node, which is the number of links it has to nodes outside its current community.

Action Plan - Detailed Steps

Sorting and Reassignment:

- Nodes are sorted based on their external degree in descending order (highest external degree first).
- For each node in this order:
  - The algorithm finds which other community has the most connections to this node.
  - If this external community has more connections to the node than its current community, the node is reassigned to this new community.

Reevaluate Community Quality:

- After all nodes are reviewed and possibly reassigned, a new **weighted graph** is created to represent the updated community assignments.
- If these changes improve the **modularity** this updated graph becomes the new best community structure.
- This process repeats until no further improvements are found within a fixed number of iterations.

Action Plan - Pheromone Trail Update in ACO

In Ant Colony Optimization (ACO), pheromone levels on paths are updated based on the iteration-best solution, which is the solution with the highest modularity in the current iteration.

Pheromone Update Formula:

$$\tau_{ij} \leftarrow \rho\tau_{ij} + \Delta\tau_{ij}$$

- τ_{ij}: Current pheromone level on edge (i, j).
- ρ: Pheromone persistence ($0 < \rho < 1$).
- $\Delta\tau_{ij}$: Amount of pheromone added to edge (i, j).

Pheromone Amount Based on Modularity:

$$\Delta\tau_{ij} = \begin{cases} M(s_{ib}) & \text{if edge (i, j) is in } s_{ib} \\ 0 & \text{otherwise} \end{cases}$$

- $M(s_{ib})$: Modularity of the iteration-best solution s_{ib}.

If edge (i, j) is part of the best solution, it receives pheromone equal to $M(s_{ib})$; otherwise, no pheromone is added.

Action Plan - Detailed Steps

Pseudocode of ant-based algorithm

- **Input:** Graph G represents social network
- **Output:** Weighted graph G_w^* whose vertices represent the community structure
- **Initialization:**
  - Initialize Ant, A, of size $|V|$
  - For each vertex $v \in V$, sort v.neighbors
  - For each vertex $v \in V$, traverse its neighbors and assign them to u
- **Exploration:**
  - While $i < i_{\max}$:
    - **ExploreGraph**(G, A)
    - **ResetAnts**(G, A)
    - Increment i
- **Construction:**
  - Sort E in decreasing order of pheromone
  - $G_w \leftarrow \text{BuildCommunities}(E)$

Pseudocode continuation

Optimization:

- $G_w^* \leftarrow \text{LocalOptimize}(G_w, G)$

Return G_w^*

Action Plan - Detailed Steps

Fuzzy Clustering and Fuzzy C-Means (FCM):

- After determining the optimal number of clusters using an ant-based algorithm, we can refine the communities with fuzzy clustering. Unlike traditional clustering, fuzzy clustering allows each data point to belong to multiple clusters, each with a certain probability.

Fuzzy C-Means (FCM) Algorithm:

- The Fuzzy C-Means (FCM) algorithm, introduced by Bezdek, aims to partition a set of data points $X = \{x_1, \ldots, x_n\}$ into K clusters $C = \{c_1, \ldots, c_K\}$. Each point is associated with all clusters through a membership degree represented in a partition matrix $U = [u_{ij}]$, where u_{ij} indicates how much point x_j belongs to cluster c_i.

Action Plan - Detailed Steps

Objective Function The goal of FCM is to minimize the following objective function:

$$J(U, V) = \sum_{i=1}^{c} \sum_{j=1}^{n} (u_{ij})^m ||x_j - v_i||^2$$

where $||x_j - v_i||$ is the distance between data point x_j and cluster center v_i, and m is the fuzziness parameter.

Steps of FCM:

1. **Initialize**: Randomly select c initial cluster centers.
2. **Calculate Membership**: Update the fuzzy membership u_{ij} using:

$$u_{ij} = 1 / \sum_{k=1}^{c} (d_{ij}/d_{ik})^{(2/m-1)}$$

where d_{ij} is the distance from point x_j to cluster center v_i and m $\in \mathbb{R},\ m > 1$.

Action Plan - Detailed Steps

Steps of FCM (Continued)

3. **Update Cluster Centers**: Compute the new fuzzy centers v_j as:

$$v_j = \frac{\sum_{i=1}^{n} (u_{ij})^m x_i}{\sum_{i=1}^{n} (u_{ij})^m} \quad \text{for } j = 1, 2, \ldots, c$$

4. **Repeat**: Continue steps 2 and 3 until J is minimized or the changes in the membership matrix U are below a certain threshold β $(0 < \beta < 1)$ i.e., at kth iteration step

$$||U^{(k+1)} - U^{(k)}|| < \beta$$

where 'U' $= (u_{ij})_{n*c}$ is the fuzzy membership matrix..

Code Implementation

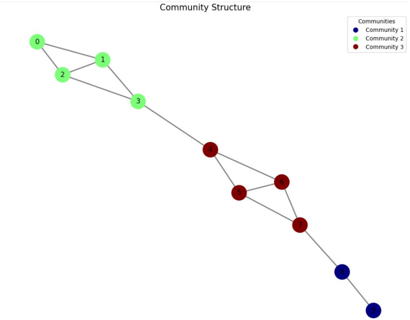
```python
def exploration_phase(adj_matrix, num_ants, alpha=1.0, beta=2.0, evaporation_rate=0.07, max_iterations=100):
    pheromone = np.ones_like(adj_matrix, dtype=float)  # Initializing pheromone matrix with all entries as 1
    ants = np.zeros(num_ants, dtype=int)  # Store the current vertex for each ant

    for i in range(num_ants):
        ants[i] = i  # Each ant starting at random position

    for iteration in range(max_iterations):
        for ant in range(num_ants):
            current_vertex = ants[ant]

            neighbors = np.where(adj_matrix[current_vertex] > 0)[0]
            if len(neighbors) == 0:
                continue  # Skip if no neighbors available
            # Calculating the probabilites
            pheromone_levels = pheromone[current_vertex, neighbors] ** alpha
            distances = adj_matrix[current_vertex, neighbors] ** (-beta)
            probabilities = pheromone_levels * distances
            probabilities /= probabilities.sum()

            # Choose the next vertex based on probabilities
            next_vertex = np.random.choice(neighbors, p=probabilities)
            ants[ant] = next_vertex  # Move the ant

            pheromone[current_vertex, next_vertex] += 1  # Increment pheromone on edge

        # Evaporate pheromones to avoid early convergence
        pheromone *= (1 - evaporation_rate)

    return pheromone
```

```python
# Construction Phase
# Construct communities based on pheromone levels
def construction_phase(pheromone, adj_matrix):
    community_structure = np.zeros(adj_matrix.shape[0], dtype=int)
    community_id = 0  # Community identifier
    community_connections = {}

    # Sort edges by pheromone levels (descending order)
    edges = [(i, j) for i in range(adj_matrix.shape[0]) for j in range(i+1, adj_matrix.shape[0])]
    edges.sort(key=lambda x: pheromone[x[0], x[1]], reverse=True)

    for i, j in edges:
        if community_structure[i] == 0 and community_structure[j] == 0:
            # Both nodes are unassigned; create a new community
            community_structure[i] = community_structure[j] = community_id
            community_id += 1
        elif community_structure[i] == 0:
            # Node i is unassigned; add it to j's community
            community_structure[i] = community_structure[j]
        elif community_structure[j] == 0:
            # Node j is unassigned; add it to i's community
            community_structure[j] = community_structure[i]
        else:
            # Both nodes belong to different communities; create/strengthen connection
            comm1 = community_structure[i]
            comm2 = community_structure[j]
            if comm1 != comm2:
                if (comm1, comm2) not in community_connections:
                    community_connections[(comm1, comm2)] = pheromone[i, j]
                else:
                    community_connections[(comm1, comm2)] += pheromone[i, j]

    return community_structure, community_connections
```

```python
# Local Optimization Phase
def optimization_phase(community_structure, adj_matrix):
    num_nodes = len(community_structure)
    internal_degree = np.zeros(num_nodes, dtype=int)
    external_degree = np.zeros(num_nodes, dtype=int)

    # Compute internal and external degrees for each node
    for i in range(num_nodes):
        for j in range(num_nodes):
            if adj_matrix[i, j] > 0:
                if np.equal(community_structure[i], community_structure[j]):
                    internal_degree[i] += 1
                else:
                    external_degree[i] += 1

    # Sort nodes by external degree in descending order
    sorted_nodes = np.argsort(-external_degree)

    # Reassign nodes to maximize community strength
    for node in sorted_nodes:
        current_community = community_structure[node]

        community_connections = {}
        for neighbor in range(num_nodes):
            if adj_matrix[node, neighbor] > 0:
                neighbor_community = community_structure[neighbor]
                if neighbor_community != current_community:
                    if neighbor_community not in community_connections:
                        community_connections[neighbor_community] = adj_matrix[node, neighbor]
                    else:
                        community_connections[neighbor_community] += adj_matrix[node, neighbor]

        # Find the community with the strongest external connection
        if community_connections:
            strongest_community = max(community_connections, key=community_connections.get)
            # Reassign node if the strongest community has more connections than the current internal degree
            if community_connections[strongest_community] > internal_degree[node]:
                community_structure[node] = strongest_community

    return community_structure
```

```python
# Fuzzy Clustering Phase (Fine-Tuning)
def fuzzy_clustering_phase(data, c, m=2, max_iter=100, epsilon=1e-6):
    n = data.shape[0]
    U = np.random.rand(n, c)
    U = U / np.sum(U, axis=1, keepdims=True)
    for iteration in range(max_iter):
        # Compute the fuzzy centers
        U_m = U ** m
        centers = (U_m.T @ data) / np.sum(U_m, axis=0)[:, None]

        # Update membership matrix
        dist = np.linalg.norm(data[:, None] - centers, axis=2)  # Shape (n, c)
        dist[dist == 0] = 1e-10


        U_new = 1.0 / np.sum((dist[:, :, None] / dist[:, None, :]) ** (2 / (m - 1)), axis=2)
        U_new = U_new / np.sum(U_new, axis=1, keepdims=True)

        # Check for convergence
        if np.linalg.norm(U_new - U) < epsilon:
            break
        U = U_new

    return U, centers
```

Output




```
Community Matrix:
   Node  Community
0     0          2
1     1          2
2     2          2
3     3          2
4     4          3
5     5          3
6     6          3
7     7          3
8     8          1
9     9          1
```

Conclusion

- The combined use of Ant Colony Optimization (ACO) and Fuzzy C-Means (FCM) allows for both efficient and precise detection of communities. ACO identifies initial connections within communities, while FCM refines these structures by assigning nodes to multiple communities with varying probabilities, enhancing accuracy.
- Tests on benchmark datasets indicate that this method outperforms traditional community detection techniques, showing competitive results compared to established approaches.
- Future improvements may focus on adapting the model to handle weighted and directed graphs, thereby broadening its practical applications.