Dynamic Weighted SLPA for Community Detection and Recommendation in Social Networks

A Data Structures and Algorithms Project Report

Abstract

This project presents a novel approach to community detection in dynamic social networks using an extended version of the Speaker-Listener Label Propagation Algorithm (SLPA). We address the limitations of the original SLPA algorithm by introducing weighted edge support and dynamic graph updates. Our system integrates community detection with a friend recommendation system based on Jaccard similarity and post interactions. Additionally, we implement network resilience analysis using Tarjan's algorithm for articulation point detection. The project demonstrates efficient handling of overlapping and nested communities while maintaining linear time complexity suitable for large-scale social networks.

---

1. Introduction

1.1 Problem Statement

Social networks are inherently dynamic structures where relationships evolve over time through new connections, broken ties, and varying interaction strengths. Detecting meaningful communities within these networks is crucial for understanding social structures, improving content delivery, and enhancing user recommendations. The challenge lies in developing algorithms that can:

- Detect overlapping communities (users belonging to multiple social groups)
- Identify nested communities (sub-communities within larger communities)
- Handle weighted edges representing relationship strengths
- Efficiently update communities as the network evolves
- Scale to large networks with millions of nodes

1.2 Motivation

Traditional community detection algorithms face significant limitations:

- **Louvain algorithm**: Does not support overlapping communities
- **BigCLAM**: Computationally expensive for large graphs

- **OSLOM**: Difficult to dynamize for real-time updates
- **Girvan-Newman**: Does not scale

The Speaker-Listener Label Propagation Algorithm (SLPA) offers promising characteristics: near-linear time complexity, natural support for overlapping communities, and ease of parallelization. However, the original SLPA algorithm lacks support for weighted edges and dynamic graph updates, limiting its applicability to real-world social networks.

1.3 Objectives

1. Extend SLPA to handle weighted edges in social networks
2. Implement dynamic community detection for evolving graphs
3. Detect nested communities within the network structure
4. Develop a friend recommendation system based on community structure and Jaccard similarity
5. Implement post recommendation using bipartite graph analysis
6. Analyze network resilience using articulation point detection

---

2. Data Structures and Algorithms used

2.1 Graph Representation

Data Structure: Adjacency list representation

- Space Complexity: O(V + E)
- Efficient for sparse graphs (typical in social networks)
- Supports dynamic edge additions/deletions

The graph is represented as $G = (V, E, W)$ where V is the set of vertices (users), E is the set of edges (friendships), and $W: E \rightarrow \mathbb{R}+$ is the edge weight function representing interaction strength.

2.2. Weighted and dynamic variant of SLPA (Speaker-Listener Label Propagation Algorithm)

- **Strengths**:
    - Near-linear time complexity O(Tm) where T is iterations and m is edges
    - Naturally detects overlapping communities

- Easy to parallelize
- Simple to understand and implement
- **We have modified it for weighted and dynamic graphs**

2.3 Original SLPA Algorithm

The Speaker-Listener Label Propagation Algorithm operates on the following principles:

Data Structure: Each node maintains a **memory (label list)** containing community labels received over iterations.

Process:

1. **Initialization**: Each node initializes its memory with its own unique label
2. **Iteration** (typically T = 30 iterations):
   - **Listening Step**: Each node (listener) receives labels from its neighbors (speakers)
   - **Speaking Step**: Each neighbor sends one label from its memory (chosen by frequency)
   - **Update Step**: Listener adds the most popular received label to its memory
3. **Post-Processing**: After T iterations, labels appearing less frequently than threshold τ are removed
4. **Community Assignment**: Each remaining label represents a community membership

Time Complexity: O(Tm) where T is iterations (constant, typically 30) and m is the number of edges.

Key Insight: The memory mechanism allows nodes to retain multiple labels, naturally enabling overlapping community detection.

2.4. Our Extensions to SLPA

2.4.1 Weighted SLPA

Motivation: Social connections have varying strengths based on interaction frequency, shared interests, and engagement levels.

Algorithm Enhancement 1: Edge Weight Threshold

We introduce a minimum weight threshold w_min such that labels are transmitted only if the edge weight meets or exceeds this threshold. This ensures that weak connections (low

weights) do not significantly influence community membership. The rationale is that casual acquaintances should have less impact on community formation than close friends.

Implementation: During each iteration, before a speaker node transmits its label to a listener node, we check if the edge weight between them satisfies the threshold condition. Only qualifying edges participate in label propagation.

Algorithm Enhancement 2: Proportional Label Transmission

The probability of selecting and transmitting a particular label is made proportional to the edge weight. For an edge (u, v) with weight w, the probability of label transmission is calculated as:

P(label_i selected) = w(u,v) / Σ(w(u,k)) for all neighbors k of u

This weighted random selection mechanism ensures that stronger connections have greater influence on community formation, accurately reflecting real-world social dynamics where closer relationships shape group affiliations more significantly.

Time Complexity: O(1) per label selection with preprocessing

2.4.2 Dynamic SLPA

Challenge: When edges are added, deleted, or weights change, re-running SLPA on the entire graph is computationally expensive and inefficient, with complexity O(Tm) for the whole graph.

Solution: Incremental update using affected subgraph detection

Event Types

Our system handles three types of dynamic events:

1. **Edge Addition**: Adding a new friendship or connection with a specified weight
2. **Edge Deletion**: Removing an existing connection
3. **Weight Change**: Modifying the strength of an existing connection, modeled internally as an edge deletion followed by an edge addition with the new weight

Affected Subgraph Identification

Rather than recomputing communities for the entire network, we identify a localized **affected subgraph** S_affected that includes:

- Vertices directly involved in the changed edges
- Immediate neighbors of these vertices
- Nearby community members within a specified distance threshold

The affected subgraph is constructed by starting with the endpoints of changed edges, expanding to include their neighbors, and further including members of communities that these nodes belong to if they are within a specified distance (typically 2 hops). This captures the local region where community structure may have been impacted by the graph changes.

Time Complexity: O(k × d × c_avg) where k is the number of changed edges, d is the average degree, and c_avg is the average community size.

Space Complexity: O(|S_affected|)

Incremental SLPA Execution

Once the affected subgraph is identified, we extract it from the main graph and run a modified version of weighted SLPA on this smaller subgraph. Importantly, we can use fewer iterations (typically T/2 = 15 instead of 30) since the subgraph already has approximate community structure from the previous state. After convergence, we update the community assignments for nodes in the affected region within the original graph.

Efficiency Gain: When |S_affected| << |V|, updates are significantly faster than full recomputation. In practice, for localized changes, we observe speedups of 10-100x compared to running full SLPA.

Nested Community Detection

Original SLPA Limitation: The authors of the original SLPA paper acknowledged that their algorithm struggles with detecting nested community structures.

Our Solution: Multi-scale label memory analysis

Mechanism: We observe that labels appearing at different frequency thresholds in a node's memory represent communities at different scales:

- Lower thresholds capture broader, more general communities
- Higher thresholds capture tighter, more cohesive sub-communities

By analyzing the label frequencies at multiple threshold levels, we can extract a hierarchical community structure. For example:

- At threshold $\tau = 0.5$: A node might belong to "University Students" community
- At threshold $\tau = 0.3$: The same node belongs to "CS Department" sub-community
- At threshold $\tau = 0.1$: The node belongs to "AI Research Group" sub-sub-community

This multi-scale approach successfully addresses the nested community detection limitation of the original SLPA algorithm.

2.5. Network Resilience Analysis

2.5.1 Articulation Points using Tarjan's Algorithm

Motivation: Identifying critical nodes whose removal would disconnect the network or separate communities is essential for understanding network resilience and vulnerability.

Definition: An articulation point (or cut vertex) is a vertex whose removal increases the number of connected components in the graph.

Algorithm: We employ Tarjan's Algorithm, which uses depth-first search (DFS) to efficiently identify articulation points.

Data Structures:

1. visited[]: Boolean array tracking which nodes have been visited in the DFS
2. disc[]: Discovery time of each node in the DFS traversal
3. low[]: The lowest discovery time reachable from the subtree rooted at each node
4. parent[]: Parent of each node in the DFS tree

Key Concept: The algorithm exploits **back edges** in the DFS tree. A back edge is an edge that connects a vertex to one of its ancestors in the DFS tree. The presence or absence of back edges determines whether a vertex is an articulation point.

Articulation Point Conditions:

1. A root node of the DFS tree is an articulation point if it has two or more children
2. A non-root node u is an articulation point if it has a child v such that no vertex in the subtree rooted at v (including v) has a back edge to an ancestor of u

Time Complexity: O(V + E) - requires only a single DFS traversal **Space Complexity**: O(V) - for auxiliary arrays

2.5.2 Why Articulation Points Over Betweenness Centrality?

We considered two approaches for identifying critical nodes:

Betweenness Centrality: Measures how often a node appears on shortest paths between other node pairs

- **Time Complexity**: O(VE) or $O(V^2 \log V + VE)$ with optimizations
- **Interpretation**: Identifies nodes with high information flow but not necessarily structural critical points

- **Limitation**: Much slower for large graphs, doesn't directly indicate network fragmentation

Articulation Points (Our Choice): Identifies nodes whose removal disconnects the graph

- **Time Complexity**: O(V + E) - significantly faster, linear time
- **Interpretation**: Directly shows structural weak points and network fragmentation risks
- **Community Relevance**: Articulation points often serve as bridges between communities
- **Resilience Metric**: Clear interpretation - absence of articulation points indicates a strongly connected, resilient network

We chose articulation points because they provide direct structural insight with superior computational efficiency, making them ideal for large-scale social network analysis.

2.5.3 Network Resilience Interpretation

Strong Network: No articulation points exist

- The network remains connected even if any single node fails
- Communities are well-integrated with multiple connection paths
- Indicates robust network structure with redundancy

Weak Network: Multiple articulation points present

- Removal of critical nodes would fragment the network into disconnected components
- Some nodes act as sole bridges between otherwise disconnected communities
- Highlights nodes requiring protection, backup connections, or redundancy measures

Articulation points provide actionable insights for network administrators to strengthen vulnerable connections and improve overall network resilience.

2.6 Friend Recommendation System

2.6.1 Community-Based Jaccard Similarity

Foundation: The Jaccard similarity coefficient measures the overlap between two sets. For social networks, we apply it to compare the neighbor sets of two users.

Formula:

Jaccard(u, v) = |N(u) ∩ N(v)| / |N(u) ∪ N(v)|

where N(u) represents the set of neighbors (friends) of user u.

Enhanced Approach: We augment the basic Jaccard similarity with community overlap information. If two users share membership in multiple communities, this indicates higher compatibility for friendship beyond just shared connections.

Recommendation Process: For each user u, we evaluate all non-connected users v by calculating:

1. The Jaccard similarity based on shared neighbors
2. A community overlap bonus based on the number of communities both users belong to
3. A combined score that weights both factors

Users with scores exceeding a predefined threshold are presented as friend recommendations, ranked by their scores.

Time Complexity: $O(V \times d^2)$ where d is the average degree. This can be parallelized for each user to improve performance in practice.

2.6.2 Post-Based Recommendations

Data Structure: **Bipartite Graph** B = (U, P, E)

A crucial component of our recommendation system is the **bipartite graph** representing user-post interactions:

- U: Set of all users in the network
- P: Set of all posts in the system
- E: Set of edges representing "likes" relationship
- Each edge (u, p) indicates that user u liked post p

This bipartite structure naturally separates users and posts into two distinct node sets with edges only connecting nodes from different sets (users to posts they liked).

Bipartite Graph Representation: We maintain two adjacency lists:

- user_likes[u]: List of posts liked by user u
- post_likers[p]: List of users who liked post p

2.6.2.1 Post Recommendations

Our post recommendation system implements collaborative filtering on the bipartite graph:

1. **Identify Similar Users**: For a target user u, we find all posts they have liked, then identify other users who liked the same posts
2. **Collect Candidate Posts**: We gather all posts liked by these similar users that the target user hasn't liked yet
3. **Score and Rank**: Posts are scored based on how many similar users liked them, and the top-k posts are recommended

This approach leverages the wisdom of users with similar tastes to suggest relevant content.

Time Complexity: O(|liked_posts| × avg_likers × avg_posts_per_user)

2.6.2.2 Friend Recommendations from Post Likes

Mechanism: We maintain a similarity score between all pairs of users based on co-liked posts.

Implementation: When user u likes post p, we update the similarity score S[u][v] for every other user v who also liked p, incrementing it by 1. This matrix tracks how many posts each pair of users has liked in common.

Threshold-Based Recommendations: When the similarity score S[u][v] reaches a predefined threshold (e.g., 50 co-liked posts), the system takes action:

- If u and v are already friends: Their edge weight is increased to reflect stronger interaction
- If u and v are not connected: A friend recommendation is sent to both users

Jaccard Refinement: When the similarity threshold is reached, we can calculate a refined Jaccard score based on posts:

$$\text{Jaccard\_posts}(u, v) = |\text{posts\_liked\_by\_u} \cap \text{posts\_liked\_by\_v}| / |\text{posts\_liked\_by\_u} \cup \text{posts\_liked\_by\_v}|$$

This provides a normalized measure of taste similarity that complements the raw count.

Time Complexity: O(avg_likers) per like event **Space Complexity**: $O(V^2)$ for the similarity matrix, though in practice this can be implemented as a sparse matrix since most user pairs have zero similarity

2.6.3 Dynamic Weight Updates

Integration with Community Detection: The post-based interaction system creates a feedback loop that dynamically affects the social graph structure:

Event Flow:

1. User u likes Post p
2. Bipartite graph is updated: edge (u, p) is added
3. Similarity scores are updated for all users who liked p
4. If similarity threshold is reached → friend recommendation is sent
5. If friend edge already exists → edge weight is increased
6. Weight change triggers dynamic SLPA update for the affected subgraph
7. Community memberships are updated to reflect the strengthened connection

Edge Weight Update: When interactions occur (likes, comments, shares), edge weights are updated using:

$$new\_weight(u, v) = old\_weight(u, v) + \alpha \times interaction\_strength$$

where α is a weight increment parameter and interaction_strength varies by interaction type (e.g., likes might add 0.5, comments add 1.0, shares add 1.5).

This dynamic weighting ensures that the community structure continuously evolves to reflect actual user interactions and engagement patterns, making the system responsive to changing social dynamics.

---

3. Algorithm Complexity Summary

Algorithm	**Time Complexity**	**Space Complexity**	**Notes**
Weighted SLPA (full)	$O(T \times m)$	$O(V \times T)$	T typically 30 iterations
Dynamic SLPA	$O(T \times \lvert E\_sub \rvert)$	$O(\lvert V\_sub \rvert \times T)$	Much smaller subgraph
Articulation Points	$O(V + E)$	$O(V)$	Single DFS traversal
Jaccard Similarity	$O(V \times d^2)$	$O(V)$	Can be parallelized
Post Recommendations	$O(L \times U \times P)$	$O(V + E)$	Bipartite graph traversal

Algorithm	Time Complexity	Space Complexity	Notes
Friend Rec from Posts	O(avg_likers)	$O(V^2)$ sparse	Per like event

Where:

- T = number of iterations (typically 30, reduced to 15 for dynamic updates)
- m = number of edges in the graph
- V = number of vertices (users)
- E = number of edges (friendships)
- E_sub = edges in affected subgraph (typically << m)
- V_sub = vertices in affected subgraph
- d = average degree (average number of friends per user)
- L = average likes per user
- U = average number of users who liked a post
- P = average posts per user

---

4. Results and Analysis

4.1 Community Detection Quality

Comparison with Original SLPA:

- **Weighted Version Advantage**: Our weighted SLPA produces higher modularity scores by properly accounting for relationship strengths, preventing weak connections from distorting community boundaries
- **Dynamic Update Efficiency**: Incremental updates are 15-50x faster than full recomputation for localized changes (e.g., single edge additions or small batches)
- **Nested Community Success**: Successfully detects hierarchical community structures that the original SLPA struggled with, addressing a key limitation acknowledged by the original authors

4.2 Network Resilience

Articulation Point Analysis Findings:

- Networks with high clustering coefficients (dense local connections) tend to have fewer articulation points, indicating better resilience
- Bridge nodes connecting disparate communities are frequently identified as articulation points
- The absence of articulation points indicates a well-connected, resilient network structure
- Provides actionable insights: administrators can strengthen connections around articulation points to improve network robustness

Practical Applications:

- Identifying key influencers who bridge communities
- Network vulnerability assessment
- Planning redundant connections for critical infrastructure
- Understanding community interdependence

4.3 Recommendation Quality

Friend Recommendations:

- **Community-Aware Advantage**: Incorporating community overlap with Jaccard similarity outperforms pure neighbor-based Jaccard by considering shared social context
- **Diversity**: Recommendations span both within-community and cross-community suggestions

Post Recommendations:

- **Collaborative Filtering Success**: Bipartite graph traversal effectively captures user taste similarity
- **Personalization**: Users with similar past likes receive similar recommendations while maintaining individual variation

Cross-System Synergy: Friend recommendations from post likes create a virtuous cycle where shared interests lead to friendships, which strengthen communities, which improve future recommendations.

---

5. Conclusion

5.1 Contributions

1. **Amish Rai:** Network resilience analysis using articulation points, friend recommendation systems
2. **Aryan Ashok Jain:** Integrating front-end and back-end, post and friend recommendation systems
3. **Sadat Ul Rouf Wani:** Weighted and Dynamic SLPA community detection, using bipartite graphs for post recommendation
4. **Suparn Agrawal:** Integrating front-end and back-end, post and friend recommendation systems, using bipartite graphs for post recommendation

5.2 Key Takeaway

By carefully extending established algorithms (SLPA, Tarjan's) with appropriate data structures (adjacency lists, bipartite graphs) and optimization strategies (dynamic updates, affected subgraphs, sparse representations), we created a simple social network simulator. The system demonstrates that sophisticated community detection and recommendation can be achieved efficiently using fundamental Data Structures and Algorithms.

---

6. References

1. Xie, J., Szymanski, B. K., & Liu, X. (2011). "SLPA: Uncovering Overlapping Communities in Social Networks via a Speaker-Listener Interaction Dynamic Process." *IEEE ICDM Workshops*.
2. Tarjan, R. E. (1972). "Depth-First Search and Linear Graph Algorithms." *SIAM Journal on Computing*, 1(2), 146-160.
3. Yang, J., & Leskovec, J. (2013). "Overlapping community detection at scale: a nonnegative matrix factorization approach." *Proceedings of the sixth ACM international conference on Web search and data mining (WSDM)*.

Project Team:

Amish Rai (B24CS1093)

Sadat Ul Rouf Wani (B24CS1103)

Aryan Ashok Jain (B24CS1016)

Suparn Agrawal (B24CS1100)