Project Report

submitted to: Arsalan Aslam

Ehtisham ul Hassan, Asad Khurshid
22I-1777,22i-1585
CYBER – B

Contents

Introduction:

In the era of social networking, understanding the structure and dynamics of social networks is pivotal for analysing relationships and influence within a community. This project focuses on leveraging graph-based algorithms to study social network data and derive meaningful insights. The dataset contains user interactions with connection weights, reflecting the intensity or strength of relationships. Through graph theory and dynamic programming, this project aims to identify significant relationships and uncover patterns of influence.

Objective:

The primary objectives of this project are:

1. To create an undirected graph representing the social network, where nodes represent users and weighted edges represent the connection strength between them.
2. To find the shortest path between any two users using Dijkstra's algorithm and A* algorithm.
3. To determine the longest chain of influence in the network using dynamic programming, based on the influence scores provided for each user.

I. Task1: Graph-Based Analysis

1. Graph Construction:

- Load the dataset containing three columns: two user IDs and the connection weight.
- Create an undirected graph using an adjacency list, where:
  - Nodes represent users.
  - Edges represent connections between users with associated weights.
- Shorter weights indicate stronger connections.

2. Shortest Path Calculation:

- **Dijkstra's Algorithm**:
  - A greedy approach to find the shortest path from a source node to a target node.
  - Uses a priority queue to process the node with the minimum distance iteratively.
- *A Algorithm** (Bonus):
  - Enhances Dijkstra's approach with a heuristic function to guide the search.
  - The heuristic h(n) is the number of direct connections (degree) of the node, reflecting its importance in the network.

- Menu-driven implementation to allow users to select the algorithm and input the source and target nodes.

3. Pseudocode:

- **Function: aStar(graph, start, goal)**
- **Input:** graph (adjacency list), start (integer), goal (integer)
  Output: path (list of nodes), cost (integer)

1. Initialize a dictionary gScore with all nodes set to infinity.
2. Set gScore[start] = 0.
3. Initialize a dictionary fScore with all nodes set to infinity.
4. Set fScore[start] = gScore[start] + heuristic(start) (number of neighbors of start).
5. Initialize a priority queue pq and push (fScore[start], start) (estimated cost, node).
6. Initialize an empty dictionary parent to track the shortest path.
7. **While** pq is not empty:
   - Pop the node with the smallest fScore (currentFScore, currentNode) from pq.
   - **If** currentNode == goal, break the loop.
   - **For each** (neighbor, weight) in graph[currentNode]:
     - Calculate tentativeGScore = gScore[currentNode] + weight.
     - **If** tentativeGScore < gScore[neighbor]:
       - Update gScore[neighbor] = tentativeGScore.
       - Update fScore[neighbor] = gScore[neighbor] + heuristic(neighbor).
       - Set parent[neighbor] = currentNode.
       - Push (fScore[neighbor], neighbor) to pq.
8. Backtrack from goal to start using parent to construct the path.
9. **Return** path and gScore[goal].

Function: main()

10. **Input:** None
    Output: None (displays results)
11. Call loadGraphFromFile(filename) to load the graph.
12. **Repeat until exit:**
13. Display:
14. "Find shortest path using Dijkstra's Algorithm"
15. "Find shortest path using A* Algorithm"

16. "Exit"
17. Get user choice.
18.

II. Dynamic Programming on Graph

1. Influence Score Data:

- Load influence scores for each user from the dataset.
- Each user is assigned a score indicating their ability to influence others.

2. Longest Increasing Path:

- Formulate a dynamic programming solution to find the longest path in the network, where:
  - Influence scores in the path strictly increase.
- For each node:
  - Check all its neighbours and recursively calculate the maximum length of the chain.

3. Pseudocode:

1. Graph Representation

Function: loadGraphFromFile(filename)

Input: File containing graph data
Output: Graph as an adjacency list

1. Create an empty graph.
2. Open the file.
3. For each line in the file:
   - Read node1, node2, and weight.
   - Add an edge between node1 and node2 with the given weight (both directions).
4. Close the file.
5. Return the graph.

2. Load Influence Scores

Function: loadInfluenceScores(filename)

Input: File containing influence scores
Output: Dictionary of users and their scores

1. Create an empty dictionary influenceScores.
2. Open the file.

3. For each line in the file:
    - Read user and score.
    - Add user and score to the dictionary.
4. Close the file.
5. Return the dictionary.

3. Find Longest Increasing Path

Function: findLongestIncreasingPath(graph, influenceScores)

Input: Graph and influence scores
Output: Maximum path length and user sequence

1. Initialize dp dictionary to store path lengths and parent dictionary to track the path.
2. Create a list of users and sort them by their influence scores (ascending).
3. Set maxLength = 0 and endNode = -1.
4. For each user in sorted list:
    - Set dp[user] = 1 (base case).
    - Set parent[user] = -1 (no parent initially).
    - For each neighbor of the user:
        - If the neighbor's influence score is less than the user's AND the path through the neighbor is longer:
            - Update dp[user].
            - Update parent[user] to the neighbor.
    - If dp[user] > maxLength:
        - Update maxLength.
        - Set endNode = user.
5. Build the path by backtracking from endNode using parent.
6. Return maxLength and the path.

4. Main Function

1. Load the graph from graphFilename.
2. Load the influence scores from influenceFilename.
3. Call findLongestIncreasingPath(graph, influenceScores).
4. Display:
    - Maximum path length.

- Sequence of users in the path.

III. Implementation

Part 1: Graph-Based Analysis

- **Input**: Dataset with user connections and weights.
- **Output**: Shortest path between two users and its cost.

Part 2: Dynamic Programming on Graph

- **Input**: Dataset with influence scores for each user.
- **Output**: Longest increasing path of influence and its length.

IV. Conclusion

This project successfully implements graph-based and dynamic programming algorithms to analyse social networks. The results demonstrate the ability to:

1. Identify significant connections and shortest paths efficiently.
2. Discover meaningful patterns of influence within the network.

The analysis provides a robust foundation for understanding complex relationships and influence dynamics in social networks. The integration of Dijkstra's algorithm, A* algorithm, and dynamic programming highlights the versatility of computational methods in analyzing large datasets.