

Problem Statement

Social networks are complex structures where users (nodes) are connected via relationships (edges). Analyzing these networks requires:

1. Efficient algorithms to find connections (shortest path, mutual friends).
2. Visualization tools to interpret network structures.
3. Interactive interfaces for non-technical users.

Challenges in Existing Solutions:

- Most tools (e.g., Gephi) are desktop-based with no Python integration.
- Lack of real-time pathfinding visualization.
- Limited customization for educational use.

Our Solution:

A Python-based GUI application that:

- Implements graph algorithms (BFS, Dijkstra's).
- Provides interactive visualization (NetworkX + Matplotlib).
- Allows theme customization and dynamic updates.

1. Introduction

The Social Network Visualizer is a Python-based application that allows users to model, analyze, and visualize social networks using graph algorithms. It provides an interactive GUI built with Tkinter and leverages NetworkX for graph operations and Matplotlib for visualization. The project focuses on implementing key Design and Analysis of Algorithms (DAA) concepts such as:

- Graph Representation (Adjacency List)
- Shortest Path Algorithms (BFS, Dijkstra's)
- Mutual Friend Detection (Common Neighbors)
- Graph Visualization (Custom Styling & Highlighting)

2. Key Features

2.1 User Management

- Users can be added with a unique User ID and Name.

- The system prevents duplicate user entries.
- Users are stored in a dictionary and represented as nodes in the graph.

2.2 Friendship Management

- Users can be connected with a friendship strength (1-10).
- The strength of a friendship is stored as an edge weight.
- The system validates if both users exist before creating an edge.

2.3 Path Finding Algorithms

I. Unweighted Shortest Path (Bidirectional BFS)

- Finds the shortest path in terms of the number of hops.
- Efficiently searches from both source and target.

II. Weighted Strongest Path (Dijkstra's Algorithm)

- Considers friendship strength as edge weights.
- Finds the path with the highest cumulative strength.

2.4 Mutual Friend Detection

- Uses `nx.common_neighbors()` to find mutual friends between two users.
- Highlights mutual friends in the visualization.

2.5 Interactive Visualization

- Dynamic Graph Rendering using Matplotlib.
- Custom Themes (Ocean, Forest, Royal) with adjustable colors.
- Highlighting for paths, mutual friends, and selected nodes.
- Edge Labels showing friendship strength.

3. Technical Implementation

3.1 Data Structures

- Graph Representation: `networkx.Graph()` (Undirected, Weighted).
- User Storage: Python dictionary (`{user_id: name}`)
- Path Finding:

- I. Bidirectional BFS (for unweighted shortest path).
- II. Dijkstra's Algorithm (for weighted strongest path).

3.2 GUI Components

Component	Purpose
Tkinter Window	Main application window
Scrollable Control Panel	User inputs & operations
Matplotlib Canvas	Interactive graph visualization
Treeview Widget	Displays user list
Theme Selector	Customizable color schemes

3.3 Algorithms Used

Algorithm	Use Case	Time Complexity
Bidirectional BFS	Shortest path (unweighted)	$O(b^{d/2})$
Dijkstra's	Strongest path (weighted)	$O((E + V) \log V)$
Common Neighbors	Mutual friends	$O(\min(\deg(u), \deg(v)))$

3.4 Technology Stack

The project is implemented using the following technologies:

- **Python:** The core programming language.
- **Tkinter:** GUI framework for user interface.
- **NetworkX:** Library for graph-based operations.
- **Matplotlib:** Used for visual representation of the network.

4. Executions

4.1 User Interface Design

- The application uses **Tkinter** to create an interactive user interface with:
 - Input fields for entering names.
 - Buttons for adding individuals and creating friendships.
 - A Matplotlib-based display for real-time graph visualization.

4.2 Graph Representation

- The social network is represented as a graph using NetworkX.
- Each person is a node, and friendships are edges between nodes.
- The network updates dynamically when new nodes or edges are added.

4.3 Data Handling and Visualization

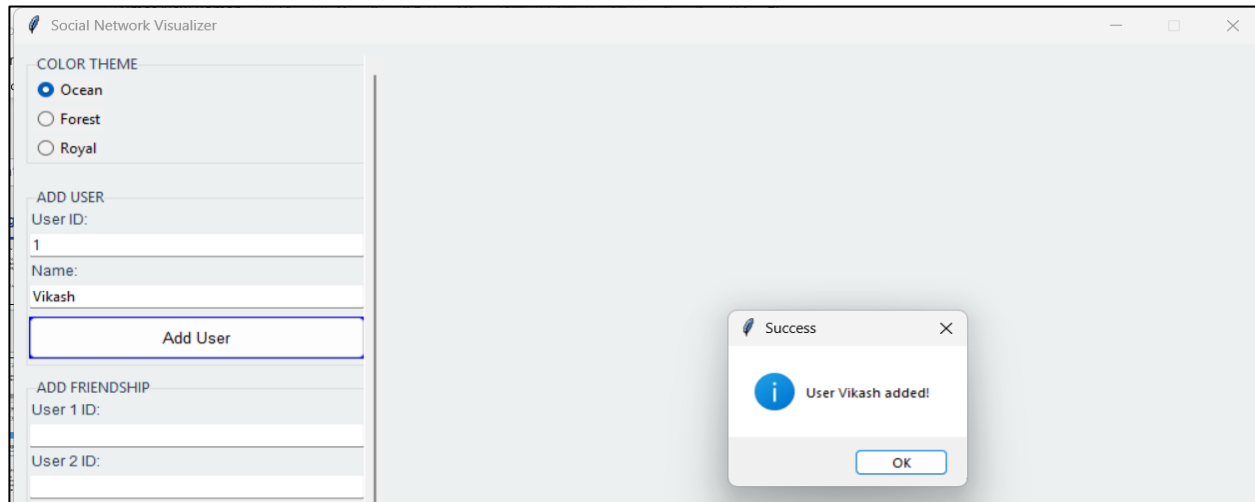
- When a user adds a person, a new node is created.
- When a friendship is added, an edge is drawn between the two nodes.
- The updated graph is displayed using Matplotlib's FigureCanvasTkAgg.

4.5 Attachments

- Please find the attachment of its execution while solve the above problem statement.



Fig 1. Main Page



Social Network Visualizer

COLOR THEME

- ☒ Ocean
- ☐ Forest
- ☐ Royal

ADD USER

User ID: 1

Name: Vikash

Add User

ADD FRIENDSHIP

User 1 ID:

User 2 ID:

Success

User Vikash added!

OK

Fig 2. Adding the user

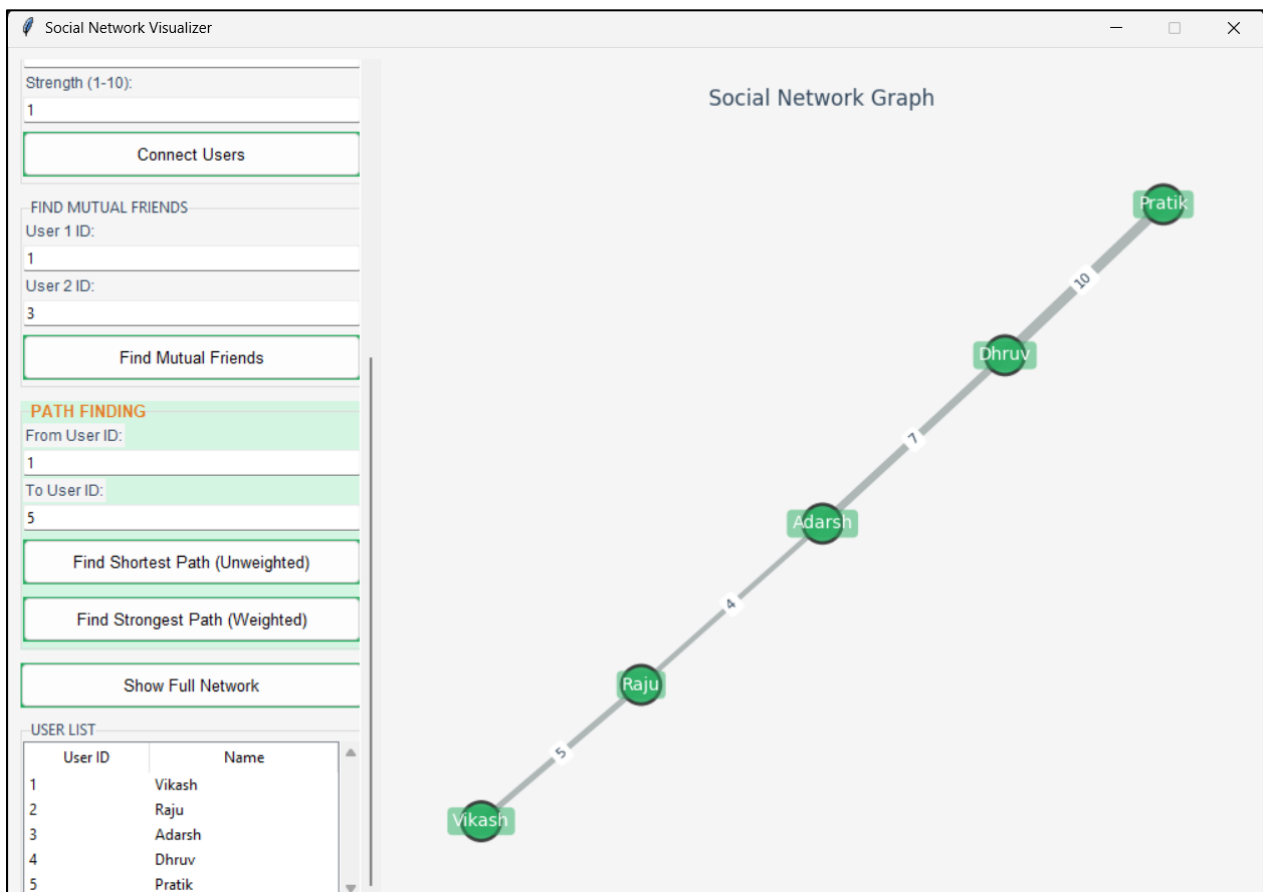


Fig 3. Full Network

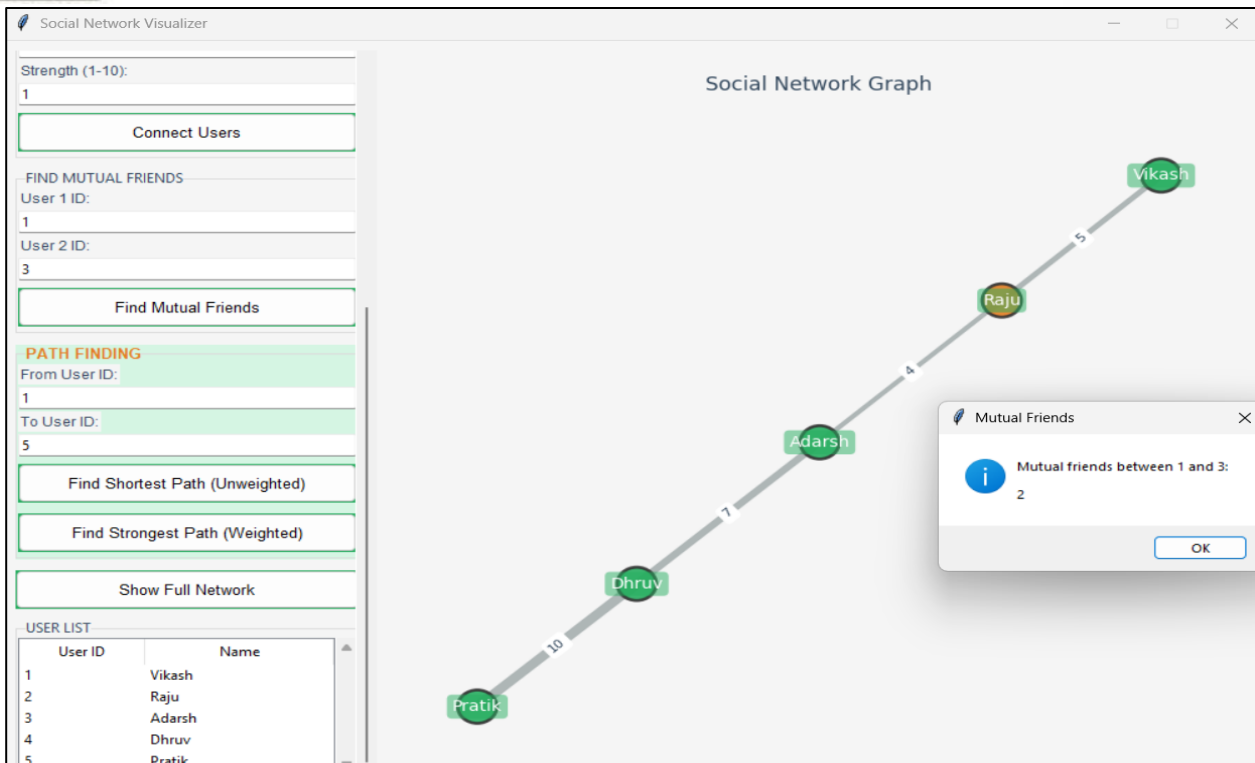


Fig 4. Mutual Friends

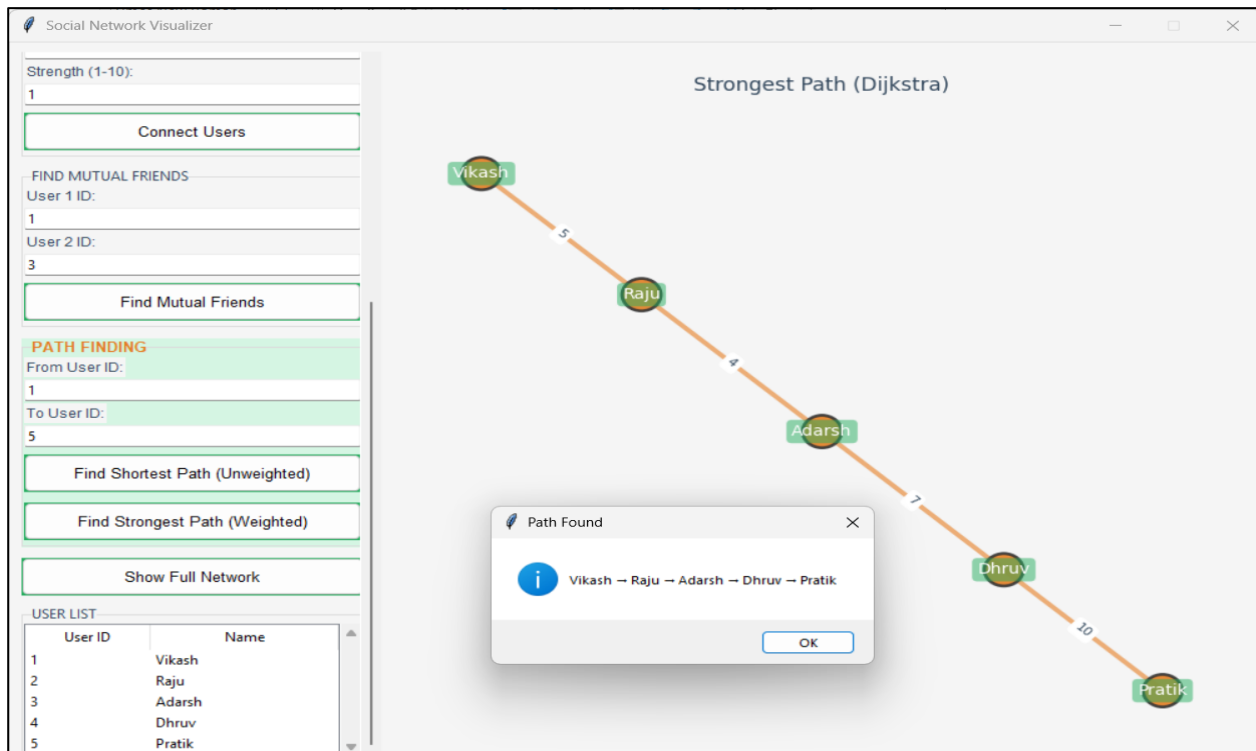


Fig 5. Shortest Path Finding (Weighted)

5. Challenges and Solutions

Limitations

- Scalability Issues
 - NetworkX visualization slows down with >100 nodes.
 - No database integration (users are stored in memory).
- Lack of Advanced Features
 - No community detection (e.g., Girvan-Newman).
 - No real-time updates (static visualization).

Solutions

- Persistence Layer
 - Store data in SQLite or Neo4j (graph database).
- Performance Optimization
 - Use PyVis for large-scale interactive graphs.
 - Implement A* for faster heuristic-based pathfinding.
- Real-Time Updates
 - Use WebSockets for dynamic network changes.
- Enhanced Features
 - Recommendation System (Friend suggestions).
 - Community Detection (Clustering algorithms).

6. Results and Benefits

- Efficient Path Finding
 - Bidirectional BFS reduces search space compared to standard BFS.
 - Dijkstra's algorithm ensures optimal weighted path detection.
- User-Friendly Interface
 - Scrollable panel for large networks.
 - Interactive visualization with zoom and pan support.
- The project serves as a foundation for more complex graph-based social analytics.

7. Future Enhancements

To further improve the application, the following features can be added:

- Database integration to store and retrieve networks.
- Graph analysis algorithms to find shortest paths, communities, or most influential nodes.
- Export functionality to save the graph as an image or dataset.
- Web application deployment via Flask/Django would enable cloud access.
- Adding multithreading support would enable real-time updates without interface lag.

8. Conclusions

The Social Network Visualizer is a powerful and interactive tool for exploring relationships in a networked environment. Built using Tkinter, NetworkX, and Matplotlib, it provides an engaging way to understand social connectivity. With potential future enhancements, it can be extended into a more comprehensive social network analysis tool.

9. References

- NetworkX Documentation – <https://networkx.org/>
- Dijkstra's Algorithm – Cormen, T. H. (2009). Introduction to Algorithms. MIT Press.
- Tkinter GUI Guide – <https://docs.python.org/3/library/tkinter.html>
- Social Network Analysis – Barabási, A. L. (2016). Network Science. Cambridge Univ. Press.
- Matplotlib Visualization – <https://matplotlib.org/>

Appendix: Project Repository

The complete source code, documentation, and installation guide for this Social Network Visualizer project are available on GitHub:

GitHub Repository:

Repository Contents:

- **main.py** - Core application code
- **requirements.txt** - Python dependencies
- **/docs** - Project documentation and screenshots