ISEASE SPREAD SIMULATION ON SOCIAL NETWORKS

Data Structures & Algorithms – Project Report

Course: Data Structures & Algorithms
Semester: Fall 2025
Date: December 21, 2025

Team Members:

- **Alap Gohar** (502082)
- **Abdullah Khalil** (501492)
- **Sikandar Hussain** (502808)

TABLE OF CONTENTS

---

1. INTRODUCTION

1.1 Overview

This project presents an interactive web-based simulation that models the spread of infectious diseases through social networks. The simulation demonstrates the practical application of **graph data structures and algorithms**, focusing on how diseases propagate through communities with realistic network topologies.

Synthetic social networks are generated using the **Barabási–Albert algorithm**, which produces scale-free networks similar to real-world social systems. These networks consist of a few highly connected individuals (hubs) and many individuals with fewer connections, closely mirroring actual social behavior.

1.2 Objectives

The primary objectives of this project are:

- Implement efficient graph data structures using adjacency lists
- Develop disease propagation algorithms using graph traversal techniques
- Create interactive visualizations of network dynamics
- Analyze algorithm complexity and performance
- Demonstrate real-world applications of data structures and algorithms

---

1.3 Technical Stack

Backend:

- Python 3.8+
- Django 5.2
- NetworkX 3.2

Frontend:

- HTML5, CSS3, JavaScript
- D3.js, Three.js
- Chart.js

---

2. SYSTEM DESIGN

2.1 Architecture

The system follows a **client–server architecture**, ensuring a clear separation between backend computation and frontend visualization. The backend is responsible for network generation and disease simulation, while the frontend handles real-time interactive visualizations.

Main Components:

- **Network Generator:** Creates scale-free networks using the Barabási–Albert algorithm
- **Disease Engine:** Simulates infection spread across the network
- **API Layer:** RESTful communication between client and server
- **Visualization Engine:** Renders interactive 2D and 3D graphs
- **Statistics Dashboard:** Displays real-time metrics and charts

---

2.2 Data Flow

The simulation begins with user-defined parameters sent to the backend. The server generates a social network and returns it in JSON format. The frontend then visualizes the network.

During simulation, infected nodes probabilistically infect neighboring nodes. After each step, the updated network state is sent back to the client, enabling real-time visualization of disease spread.

---

3. DATA STRUCTURES AND ALGORITHMS

3.1 Graph Data Structure

The core data structure is an **undirected graph** implemented using an **adjacency list**, which is optimal for sparse social networks.

Space Complexity:

- **O(V + E)**, where *V* represents vertices and *E* represents edges

Comparison:

- 1000 nodes (average degree = 6): ~12 KB using adjacency lists
- Adjacency matrix: ~1 MB for the same network

Each node stores the following attributes using hash tables:

- `infected` (Boolean)
- `infection_time` (Integer)
- `degree` (Number of connections)

To improve efficiency, adjacency lists are cached, allowing constant-time neighbor access during simulation.

3.2 Barabási–Albert Network Generation

The Barabási–Albert algorithm produces scale-free networks using **preferential attachment**, where new nodes are more likely to connect to high-degree nodes.

Algorithm Steps:

1. Initialize a complete graph with *m* nodes
2. For each new node from *m* to *n*:

- Select *m* existing nodes proportional to their degree
- Create edges to selected nodes
- Update degree counts

Time Complexity: O(V × m)
Space Complexity: O(V + E)

For typical parameters (*n* = 1000, *m* = 3), network generation completes in approximately **0.5 seconds**.

3.3 Disease Propagation Algorithm

Disease spread is modeled using a **modified Breadth-First Search (BFS)** approach. All infected nodes at time *t* attempt to infect their neighbors at time *t + 1*.

Algorithm Steps:

1. Identify currently infected nodes
2. For each infected node:
    - Retrieve neighboring nodes
    - For each susceptible neighbor:
        - Generate a random number (0–1)
        - If the number is less than infection probability, infect the neighbor
3. Update infection statistics

Time Complexity: O(V + E) per simulation step
Space Complexity: O(V)

The algorithm uses **Monte Carlo simulation**, treating each contact as an independent Bernoulli trial.

3.4 Force-Directed Layout

Visualization is implemented using a **force-directed layout**, where:

- Nodes repel each other
- Edges act like springs

The **Barnes–Hut optimization** reduces complexity from $O(V^2)$ to **O(V log V)**, enabling smooth rendering at **60 FPS** for networks up to 1000 nodes.

---

4. IMPLEMENTATION

4.1 File Structure

The project is organized as a Django application:

- `algorithms/network_generator.py` – Network generation (≈100 lines)
- `algorithms/disease_engine.py` – Disease propagation (≈150 lines)
- `views.py` – API endpoints (≈300 lines)
- `static/js/main.js` – Visualization logic (1000+ lines)
- `templates/index.html` – User interface

4.2 Key Features

Network Generation

- Configurable network size (100–2000 nodes)
- Adjustable connectivity
- Automatic network metrics

Simulation Controls

- Variable infection probability
- Initial infection selection
- Step-by-step or automatic simulation

Visualization

- 2D (D3.js) and 3D (Three.js) modes
- Color-coded nodes (Green = Healthy, Red = Infected)
- Zoom, pan, and hover interactions

Analytics

- Real-time infection statistics
- Time-series infection charts
- Degree distribution analysis

4.3 Optimization Techniques

- **Adjacency List Caching:** 3× faster simulation
- **Python List Comprehensions:** Reduced loop overhead

- **Frontend Debouncing:** Maintains 60 FPS
- **Efficient JSON Serialization:** O(V + E) complexity

---

5. RESULTS AND PERFORMANCE

5.1 Performance Benchmarks

Network Size	Generation Time	Step Time	Memory	FPS
100 nodes	0.05 s	0.002 s	2 MB	60
500 nodes	0.25 s	0.008 s	4 MB	60
1000 nodes	0.48 s	0.015 s	8 MB	60
2000 nodes	1.20 s	0.030 s	15 MB	45–55

Performance scales linearly, validating theoretical complexity predictions.

5.2 Epidemic Dynamics

- **Low Transmission (p = 0.1):** Localized outbreak (15%)
- **Medium Transmission (p = 0.3):** Moderate epidemic (65%)
- **High Transmission (p = 0.6):** Rapid pandemic (95%)

5.3 Hub Impact Analysis

Infections starting from highly connected hubs spread **twice as fast** compared to random nodes, highlighting the role of hubs as super-spreaders.

5.4 Network Properties

- Power-law degree distribution
- Small-world property (avg. path length ≈ 4.2)
- Single connected component

---

6. CONCLUSION

6.1 Summary

This project demonstrates the effective use of **graph data structures and BFS-based algorithms** for modeling disease spread. Real-time performance was achieved for networks up to 2000 nodes, with accurate complexity validation.

6.2 Key Achievements

- Implemented Barabási–Albert algorithm
- Designed efficient adjacency-list graph structure
- Developed probabilistic BFS disease simulation
- Achieved real-time visualization

6.3 Learning Outcomes

- Graph representation trade-offs
- Algorithm complexity analysis
- Probabilistic modeling
- Full-stack integration

6.4 Future Enhancements

- Vaccination and intervention modeling
- Alternative network topologies
- Parallel and GPU acceleration

6.5 Applications

- Social media information spread
- Viral marketing
- Network failure analysis
- Supply chain modeling

---

7. REFERENCES

Barabási, A.-L., & Albert, R. (1999). *Emergence of Scaling in Random Networks*. Science.
Newman, M. E. J. (2002). *Spread of Epidemic Disease on Networks*. Physical Review E.
Cormen et al. (2009). *Introduction to Algorithms*. MIT Press.
Barabási, A.-L. (2016). *Network Science*. Cambridge University Press.