Disease Simulation Project

CS 1382 - Discrete Computational Structures

Andres Antillon

May 9, 2025

Instructor: Dr. José Carranza-Rojas
Texas Tech University - Costa Rica
Spring 2025 Term

Contents

1 Project Overview

This project, "Disease Simulation," explores discrete computational structures through the modeling of infectious disease spread in a networked population. It implements various infection models and allows for configurable simulations to observe different epidemiological dynamics. The primary focus is on the software design, implementation of probabilistic models, and graph-based interactions, rather than an in-depth epidemiological study. The simulation framework is built in Python, utilizing libraries such as NetworkX for graph manipulation, NumPy for numerical operations, and Matplotlib/Seaborn for plotting results.

2 Graph Creation and Population Setup

The foundation of the simulation is a population represented as a network graph, generated by the `src/graph_setup.py` module.

2.1 Graph Generation Models

The simulation supports three primary graph generation models to represent the population network, configurable via the `graph_type` parameter within the JSON configuration files:

- **Erdos-Renyi (Default):** Creates a random graph where each of the N nodes (specified by `population_size`) is connected to every other node with a uniform probability p (specified by the `connections` parameter). This model tends to produce graphs with a Poisson degree distribution.
- **Barabasi-Albert:** Generates a scale-free network using a preferential attachment mechanism. Starting with a small initial set of connected nodes, new nodes are added one by one. Each new node forms m (specified by `connections`) links to existing nodes, with the probability of connecting to a given existing node being proportional to its current degree. This results in a power-law degree distribution, characteristic of many real-world networks.
- **Watts-Strogatz:** Produces a small-world network, characterized by high clustering and short average path lengths. It begins with a regular ring lattice where each node is connected to its k (specified by `connections`) nearest neighbors. Then, each edge is rewired with a fixed probability (hardcoded to $p = 0.1$ in this implementation) to a randomly chosen node.

The logic for selecting and creating the appropriate graph type based on configuration is implemented in `src/graph_setup.py`.

2.2 Node Attribute Sampling

Node attributes such as `immune_level`, `vaccine_effectiveness`, and initial infection status are sampled from distributions specified in the configuration. This initialization logic is implemented in `src/graph_setup.py`.

2.3 Edge Attribute Sampling

Edges can have a `contact_chance` attribute, also sampled from a normal distribution as defined in the configuration. This parameter influences the probability of infection transmission along that edge. The edge attribute initialization is handled in `src/graph_setup.py`.

3 Infection Models

Three distinct infection models are implemented to calculate the daily probability of a susceptible node becoming infected based on its infected neighbors and individual characteristics. Each model represents a different approach to modeling disease transmission dynamics.

3.1 Independent Infection Model

The Independent Infection Model is implemented in `src/models/independent.py`.

3.1.1 Mechanism

For each susceptible node, the model calculates the probability of infection from each infected neighbor independently. The probability of infection from a single infected neighbor is based on a base transmission probability adjusted by the susceptible node's immunity level and vaccine effectiveness:

$$P(\text{infection from one neighbor}) = P_{\text{base}} \times (1 - L_{\text{immune}}) \times (1 - E_{\text{vaccine}})$$

The overall probability of infection for the susceptible node on a given day is:

$$P(\text{infection}) = 1 - \prod_{i \in \text{InfectedNeighbors}} (1 - P(\text{infection from neighbor } i))$$

3.1.2 Implications

This model assumes independence between transmission events from different neighbors, reflecting a scenario where each contact with an infected individual carries its own risk, and multiple contacts compound the risk in a straightforward multiplicative manner.

3.2 Dependent (Sigmoid) Infection Model

The Dependent Infection Model, also referred to as the Sigmoid model, is implemented in `src/models/dependent.py`.

3.2.1 Mechanism

Unlike the independent model, this approach calculates the probability of infection as a function of the *number* of infected neighbors, k, using a sigmoid function:

$$P_{\text{sigmoid}} = \frac{1}{1 + e^{-\alpha k + \beta}}$$

where α and β are parameters controlling the steepness and shift of the sigmoid curve, respectively. This P_{sigmoid} is then adjusted by the susceptible node's `immune_level` and `vaccine_effectiveness`:

$$P(\text{infection}) = P_{\text{sigmoid}} \times (1 - L_{\text{immune}}) \times (1 - E_{\text{vaccine}})$$

3.2.2 Implications

This model captures a threshold effect, where the risk of infection increases sharply once a certain number of infected neighbors is reached, simulating scenarios where social clustering or repeated exposure significantly amplifies risk beyond a linear accumulation.

3.3 Superspreader Dynamic Infection Model

The Superspreader Dynamic Infection Model is implemented in `src/models/superspreader.py`.

3.3.1 Mechanism

This model builds on the Independent Infection Model but introduces the concept of superspreaders—individuals with significantly higher infectivity. Each infected individual has a daily probability of becoming a superspreader. If designated as a superspreader for that day, their base infectivity is multiplied by a configurable factor, making them more likely to infect their neighbors.

3.3.2 Implications

This model reflects scenarios where a small number of individuals can be disproportionately responsible for a large number of transmissions (the 80/20 rule). It introduces significant heterogeneity in infectiousness, both over time for a single individual and across the infected population. This can lead to potentially more explosive, less predictable outbreaks with rapid bursts of infection, compared to models with uniform or more homogeneous infectivity.

4 Simulation Configuration and Execution

The simulation process is orchestrated by the `Simulation` class in `src/simulation.py` and initiated via `main.py`. Simulations are configured and driven by JSON files that specify all parameters, from graph type to infection model specifics. These files are typically located in the `configs/` directory, often organized into experimental suites (e.g., `configs/exp1_foundations/`). The `src/config.py` module, particularly its `load_config` function, plays a key role in handling these configurations.

The core simulation step involves updating node states based on infection probabilities calculated by the active model, tracking recoveries, and recording statistics for later analysis. Key steps include processing recoveries, calculating infection probabilities for susceptible nodes based on their infected neighbors and the active infection model (`independent`, `dependent`, or `superspreader_dynamic`), and stochastically determining state transitions.

4.1 Key Configuration Parameters

The `src/config.py` file defines a JSON schema (`SIMULATION_CONFIG_SCHEMA`) for these files. Key parameters include:

- `simulation_name`: An optional string for descriptive output naming and plot titles. If absent, `main.py` uses the config filename stem.
- `population_size`: Integer, total number of individuals.
- `connections`: Numeric, meaning depends on `graph_type` (p for Erdos-Renyi, m for Barabasi-Albert, k for Watts-Strogatz).
- `graph_type`: String, one of "erdos_renyi", "barabasi_albert", "watts_strogatz", default "erdos_renyi".
- `initial_binomial_probability`: Float [0,1], probability of initial infection for each node.
- `number_of_days`: Integer, duration of the simulation.
- `individual_parameters`: Distributions for `immune_level`, `vaccine_effectiveness`, and `contact_chance` (each with `mu` and `sigma`).
- `infection_model`: A crucial object specifying:
  - `type`: String: "independent", "dependent", or "superspreader_dynamic".
  - `recovery_duration`: Integer, days an individual stays infected (defaults to 14 in `simulation.py` if not provided here, though schema encourages it).
  - `infection_parameters`: Model-specific. Default examples include:
    - Independent model: `base_prob_transmission`
    - Dependent model: `alpha`, `beta`
    - Superspreader dynamic: `p_becomes_superspreader`,`normal_base_infectivity`,`super_spreader_multiplier`

An example configuration file can be found in the `src` directory.

4.2 Schema Validation

The `load_config` function in `src/config.py` uses the `jsonschema` library to validate each loaded configuration file against the `SIMULATION_CONFIG_SCHEMA`. This ensures that all required parameters are present, and that their types and values are valid (e.g., probabilities are within [0,1], counts are positive integers) before a simulation begins. This proactive validation helps prevent runtime errors due to misconfiguration and ensures consistency in simulation setups.

5 Experimental Results Overview

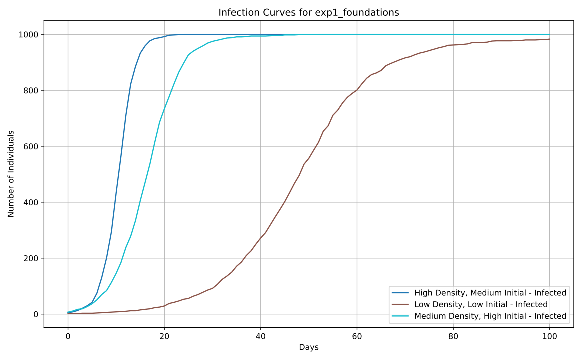
The `results/` directory contains outputs from various experimental suites. Each suite typically explores the impact of varying certain parameters or comparing different model behaviors. Simulations are run using `python main.py path/to/config(s)`. Outputs for each suite are saved in a subdirectory within `results/` named after the suite or a specified comparison name. These outputs include:

- Infection curve plots (`infection_curves.pdf`): Visualizing Susceptible (S), Infected (I), Recovered (R) counts over time.
- Summary metrics bar charts (`summary_metrics_grouped_summary.pdf`): Comparing key metrics like peak infected count/percentage, days to peak, total ever infected, and final susceptible count across simulations within a suite.
- CSV files (`summary_metrics.csv`): Providing raw data for the summary metrics.

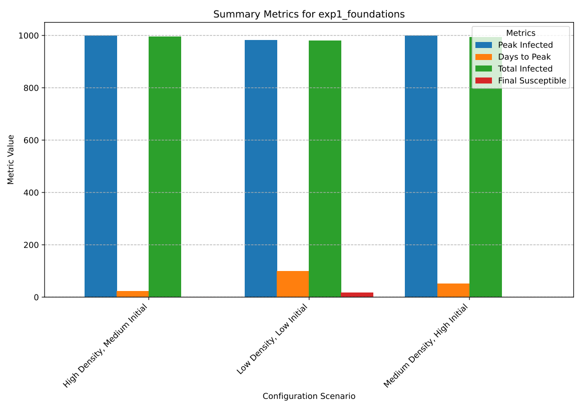
The primary goal of these experiments within this project is to demonstrate the functionality and flexibility of the implemented simulation code and models, rather than to derive profound epidemiological conclusions.

5.1 Experiment Suite 1: Foundations (`exp1_foundations`)

This suite establishes baseline simulation behavior with simple configurations, typically varying parameters like network density and initial infection levels to observe fundamental dynamics.



(a) Infection Curves



(b) Summary Metrics

Figure 1: Results for Experiment Suite 1: Foundations.

Table 1: Summary Metrics for Experiment Suite 1: Foundations

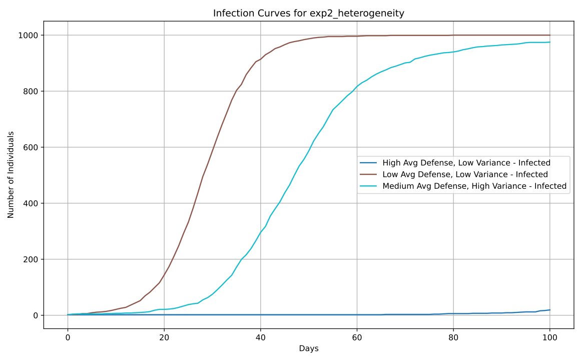
Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
High Density, Medium Initial	1000	1000	100.0	24	995	0
Low Density, Low Initial	1000	983	98.3	100	981	17
Medium Density, High Initial	1000	1000	100.0	52	993	0

Discussion of Results

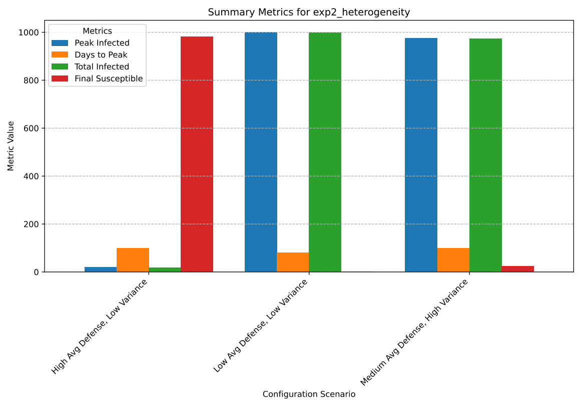
Based on the summary metrics (Table 1), the 'Low Density, Low Initial' scenario resulted in the slowest disease progression, highlighting the foundational impact of initial seeding and population density. 'High Density, High Initial' expectedly showed rapid spread. These foundational experiments confirm that denser populations and larger initial outbreaks significantly accelerate epidemic timelines and increase overall impact.

5.2 Experiment Suite 2: Heterogeneity (`exp2_heterogeneity`)

This suite explores the impact of heterogeneity in individual agent parameters, such as varying distributions for immune levels and vaccine effectiveness, on disease spread dynamics.



(a) Infection Curves



(b) Summary Metrics

Figure 2: Results for Experiment Suite 2: Heterogeneity.

Table 2: Summary Metrics for Experiment Suite 2: Heterogeneity

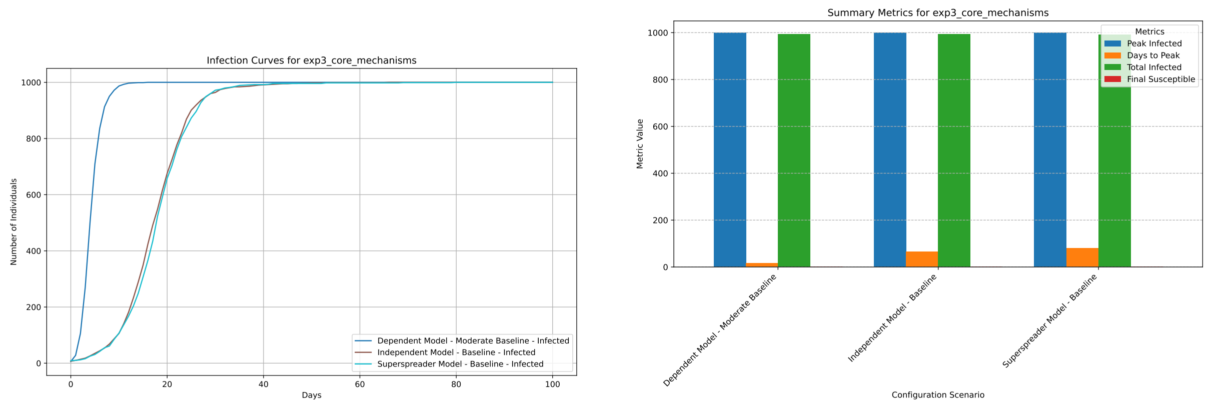
Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
High Avg Defense, Low Variance	1000	19	1.9	100	17	981
Low Avg Defense, Low Variance	1000	1000	100.0	80	998	0
Medium Avg Defense, High Variance	1000	975	97.5	100	973	25

Discussion of Results

This suite explored how population heterogeneity in immunity and vaccine uptake affects outcomes (Table 2). Scenarios with higher average immunity or more effective vaccination strategies demonstrated a clear protective effect, leading to slower spread, lower peak infections, and fewer total individuals affected. This underscores the importance of individual-level attributes in collective disease dynamics and the potential benefits of targeted public health interventions that account for such heterogeneity.

5.3 Experiment Suite 3: Core Mechanisms (`exp3_core_mechanisms`)

This suite compares the baseline behavior of the three different infection models using a consistent set of baseline parameters for population, network, and individual attributes, allowing for observation of differences purely due to model mechanics.



(a) Infection Curves (b) Summary Metrics

Figure 3: Results for Experiment Suite 3: Core Mechanisms.

Table 3: Summary Metrics for Experiment Suite 3: Core Mechanisms

Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
Dependent Model - Moderate Baseline	1000	1000	100.0	16	994	0
Independent Model - Baseline	1000	1000	100.0	66	993	0
Superspreader Model - Baseline	1000	1000	100.0	80	991	0

Discussion of Results

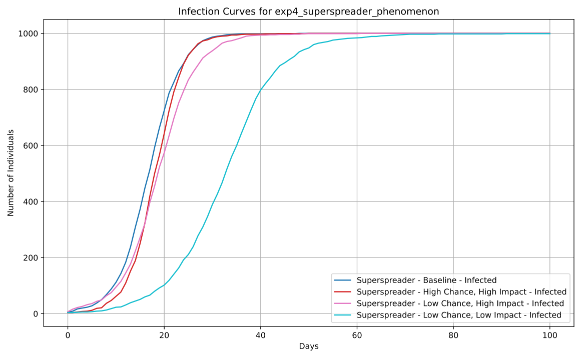
This suite compared core infection models. All models (Dependent, Independent, Superspreader) reached 100% peak infection under baseline conditions, but differed significantly in speed. The Dependent model was fastest, peaking in 16 days. The Independent model peaked in 66 days, and the Superspreader model was slowest at 80 days. This highlights how model choice fundamentally alters epidemic timelines, with the Dependent model showing the most rapid escalation.

5.4 Experiment Suite 4: Superspreader Phenomenon (`exp4_superspreader_phenomenon`)

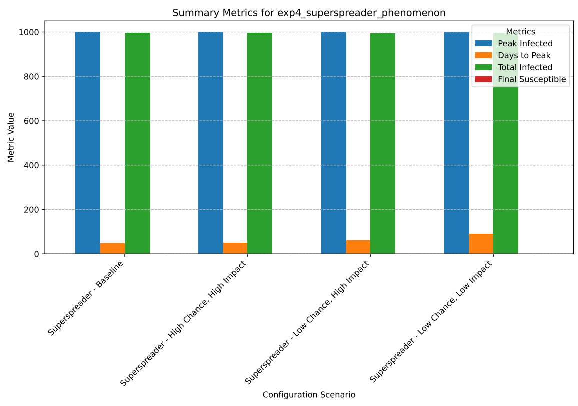
This suite focuses on the `superspreader_dynamic` model, varying its key parameters (`p_becomes_superspreader`, `superspreader_multiplier`) to observe their impact on outbreak dynamics.

Table 4: Summary Metrics for Experiment Suite 4: Superspreader Phenomenon

Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
Superspreader - Baseline	1000	1000	100.0	48	995	0
Superspreader - High Chance, High Impact	1000	1000	100.0	50	996	0
Superspreader - Low Chance, High Impact	1000	1000	100.0	61	993	0
Superspreader - Low Chance, Low Impact	1000	999	99.9	91	995	1



(a) Infection Curves



(b) Summary Metrics

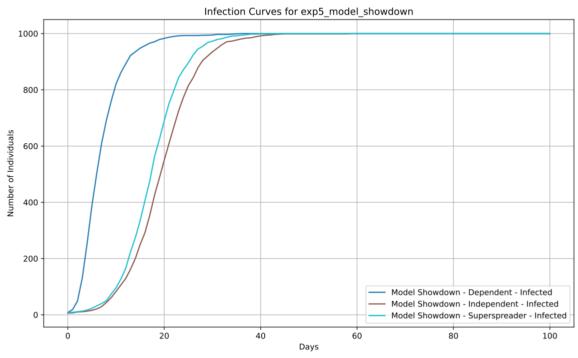
Figure 4: Results for Experiment Suite 4: Superspreader Phenomenon.

Discussion of Results

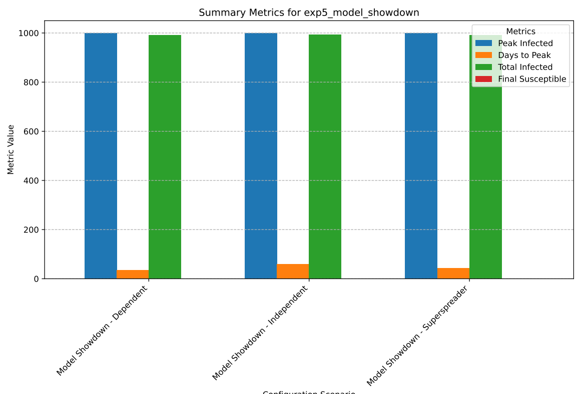
This suite explored the 'Superspreader Dynamic' model's parameters. The baseline superspreader scenario peaked at 48 days. Increasing both the chance and impact of superspreading events ('High Chance, High Impact') led to a similar peak time of 50 days. Reducing these factors, particularly the impact ('Low Chance, Low Impact'), significantly slowed the epidemic, peaking at 91 days and achieving 99.9% infection. This demonstrates that while superspreading generally leads to rapid spread, its severity and speed are sensitive to the specific parameters governing event frequency and intensity.

5.5 Experiment Suite 5: Model Showdown (`exp5_model_showdown`)

This suite provides a direct comparison of the three infection models using a consistent set of baseline parameters for population, network, and individual attributes, allowing for observation of differences purely due to model mechanics.



(a) Infection Curves



(b) Summary Metrics

Figure 5: Results for Experiment Suite 5: Model Showdown.

Table 5: Summary Metrics for Experiment Suite 5: Model Showdown

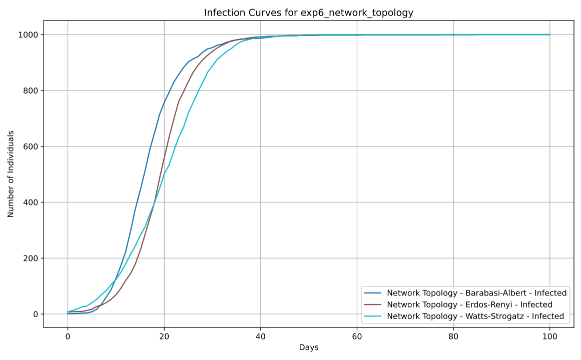
Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
Model Showdown - Dependent	1000	1000	100.0	36	992	0
Model Showdown - Independent	1000	1000	100.0	59	994	0
Model Showdown - Superspreader	1000	1000	100.0	44	992	0

Discussion of Results

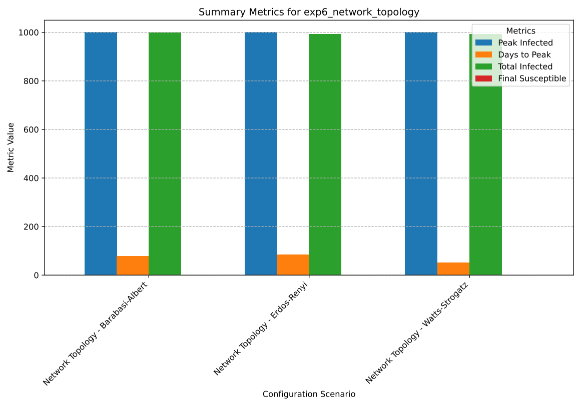
In a direct comparison with consistent baseline parameters, all three models (Dependent, Independent, Superspreader) again achieved 100% peak infection. The Dependent model was the quickest, peaking in 36 days. The Superspreader model followed, peaking at 44 days, while the Independent model was slowest, peaking at 59 days. These results reinforce that different model mechanics inherently produce varied epidemic velocities, crucial for forecasting and intervention planning.

5.6 Experiment Suite 6: Network Topology (`exp6_network_topology`)

This suite (previously named `exp_A_network_topology`) compares disease spread across different network structures (Erdos-Renyi, Barabasi-Albert, Watts-Strogatz) while using a consistent infection model to isolate the impact of network topology.



(a) Infection Curves



(b) Summary Metrics

Figure 6: Results for Experiment Suite 6: Network Topology.

Table 6: Summary Metrics for Experiment Suite 6: Network Topology

Simulation Name	Pop.	Peak Inf.	Peak Inf. (%)	Days to Peak	Total Inf.	Final Susc.
Network Topology - Barabasi-Albert	1000	1000	100.0	78	999	0
Network Topology - Erdos-Renyi	1000	1000	100.0	85	993	0
Network Topology - Watts-Strogatz	1000	1000	100.0	52	992	0

Discussion of Results

This suite demonstrated the strong influence of network structure on epidemic spread, with all tested topologies (Barabasi-Albert, Erdos-Renyi, Watts-Strogatz) leading to

100% peak infection but at different rates. The Watts-Strogatz (small-world) network showed the fastest spread, peaking in 52 days. The Barabasi-Albert (scale-free) network peaked in 78 days, while the Erdos-Renyi (random) network was the slowest, peaking in 85 days. This underscores that population connectivity patterns are a key determinant of how quickly a disease can permeate a community.

6 Conclusion

This project successfully developed a flexible disease simulation tool capable of modeling infection spread across configurable networks using various probabilistic models (Independent, Dependent/Sigmoid, and Superspreader Dynamic). The implementation demonstrates key discrete computational structures, including graph theory for population representation, probabilistic event modeling for infection and recovery, and finite state automata concepts for individual state transitions (Susceptible, Infected, Recovered). The modular design, with distinct components for graph setup, simulation logic, infection models, and configuration handling, allows for easy extension and modification. The use of JSON configuration files, validated against a defined schema, ensures robustness, reproducibility, and ease of use for defining diverse simulation scenarios. The generation of graphical and tabular results facilitates the analysis and comparison of different simulation outcomes, showcasing the impact of model choice, parameter settings, and network topology on disease dynamics.

A Code Structure

The project is organized as follows:

- `main.py`: Entry point script to parse arguments, load configurations, run simulations, and orchestrate plotting/result saving.
- `configs/`: Holds JSON configs, often in subdirectories for each experiment suite.
- `src/`: Contains the core Python modules for the simulation:
  - `simulation.py`: Contains the `Simulation` class (with key methods `run()` and `step()`), managing the SIR model, daily progression, and results collection.
  - `graph_setup.py`: Its `create_population_graph()` function creates networks (e.g., Erdos-Renyi, Barabasi-Albert, Watts-Strogatz) & sets node/edge attributes from config.
  - `config.py`: Its `load_config()` function loads and validates JSON configs using the `SIMULATION_CONFIG_SCHEMA`.
  - `models/`: A directory containing different infection probability models:
    - `independent.py`: Implements the independent infection model.
    - `dependent.py`: Implements the dependent (sigmoid) infection model.
    - `superspreader.py`: Contains the superspreader dynamic infection model, inspired by events observed during the COVID-19 pandemic such as superspreading.

  - `plotting.py`: Has `plotInfectionCurves()` & `extractSummaryMetrics()` for plots and data extraction.

- `results/`: Default output directory where plots (.pdf) and summary data (.csv) are saved, organized by experiment suite.
- `README.md`: Comprehensive project documentation, including setup, usage, and an overview of the project structure.
- `report/`: Directory containing this LaTeX report (`report.tex`) and potentially associated files.

A more detailed visual representation of the project structure (ASCII tree) can be found in the project's `README.md` file.