Network Dynamics, and Learning (Homework 3)

Hesam Khanjani 310141
Shaghayegh Abedi 309894
Nastaran Ahmadi Bonakdar 310073
Mohammadreza Mashhadigholamali 308499
Ali Samimi Fard 308956

January 21, 2024

Contributions to the Project

Question one is solved by Hesam Khanjani with the help of Ali Samimi Fard.

Question two is solved by Hesam Khanjani and Mohammadreza Mashhadigholamali

Question three is solved by Shaghayegh Abedi and Nastaran Ahmadi Bonakdar.

Question four is solved by Shaghayegh Abedi and Nastaran Ahmadi Bonakdar.

Coloring is solved by Ali Samimi Fard and Mohammadreza Mashhadigholamali with the help of Hesam Khanjani.

The report was written in LaTeX by Hesam Khanjani, Mohammadreza Mashhadigholamali, and Ali Samimi Fard.

1 Influenza H1N1 2009 Pandemic in Sweden

1.1 Epidemic on a Known Graph

In the first part of our exploration, we want to delve into the idea of simulating how diseases spread on a certain type of patterned network. We imagine a group of people connected to each other, where each person has exactly four friends. To achieve this, we create a function to simulate the epidemic, and for the input, we can choose the number of persons as *node_count*, infection rate as *infection_rate*, recovery rate as *recovery_rate*, the number of initially infected persons as *initial_infected_count*, and the total number of weeks as *total_weeks*.

These connections form what we call a symmetric k-regular undirected graph. Individuals in this network can be in one of three states: healthy (**S**), infected (**I**), or recovered (**R**).

- β: This is like the chance that a sick person passes the disease to a healthy person.
- ρ: This is the likelihood of someone getting better after being sick.
- m: It's the number of sick friends a person has.

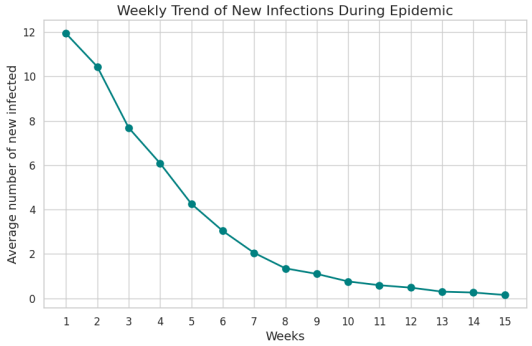
The formula that used for epidemic is driven by the following transition probabilities.

$$P(X_i(t+1) = I | X_i(t) = S, \sum_{j \in V} W_{ij} \delta_{X_j(t)=m}) = 1 - (1-\beta)^m \tag{1}$$

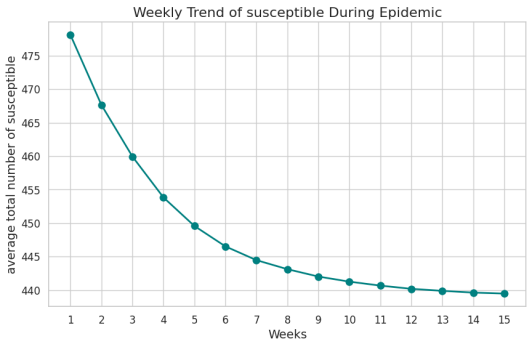
$$P(X_i(t+1) = R | X_i(t) = I) = \rho \tag{2}$$

where $\sum_{j \in V} W_{ij} \delta_{X_j(t)}$ is the number of infected neighbors for node i.

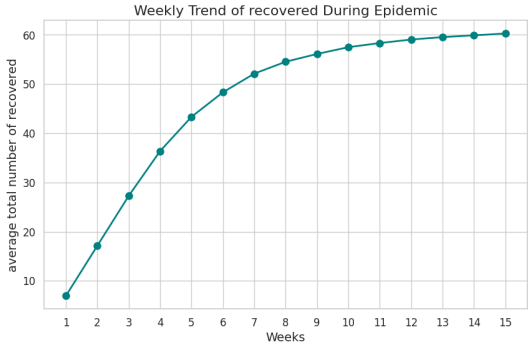
Now, we're going to simulate this on a specific network with 500 people, each having four friends, and we'll track what happens over 15 weeks. We start with 10 people already sick. The goal is to see how many new people get sick each week and how many people are in each state (healthy, sick, or recovered) on average. We tried to simulate this epidemic 100 times for each week and you see the result of this simulationin in figure 1. Also you can see comparable results in in figure 2



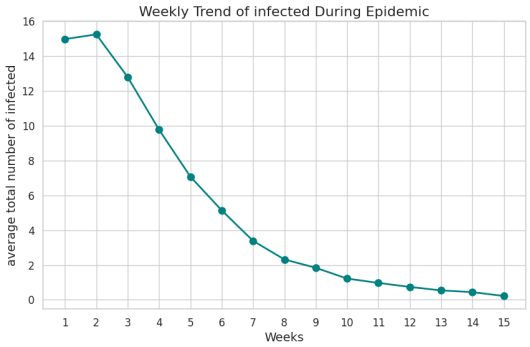
(a)



(b)



(c)



(d)

Figure 1

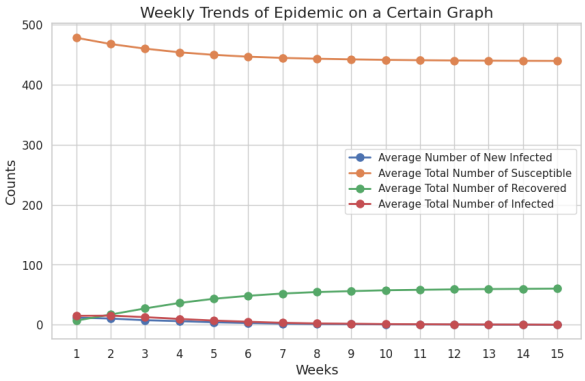


Figure 2

1.2 Epidemic on a Random Graph

The fundamental idea is that new nodes entering the network are more likely to connect to existing nodes that already have a high degree (number of connections). This concept captures the notion of "the rich get richer," where popular nodes tend to attract more links.

In the code, I've implemented a function named create preferential attachment graph that generates a graph following the Preferential Attachment model. We start with a complete graph on k+1 nodes, where k is a parameter indicating the average degree of each node. From node k+1 onwards, we iteratively add new nodes to the graph. Each new node connects to existing nodes based on a probability distribution determined by the degrees of the current nodes. The probability of connecting to a particular existing node is proportional to its degree – nodes with higher degrees are more likely to be selected. To control the number of links each new node forms, we use a mechanism that alternates

between rounding up and rounding down the desired number of links. This is done to ensure that the total number of links remains roughly constant over time. For each new node, we randomly select a subset of existing nodes to connect to, based on the calculated probability distribution. We add edges between the new node and its chosen neighbors. The resulting graph exhibits the preferential attachment phenomenon, where a few nodes become highly connected while most nodes have relatively few connections. Finally we tried to visualize this random graph with 900 as you can se in figure3

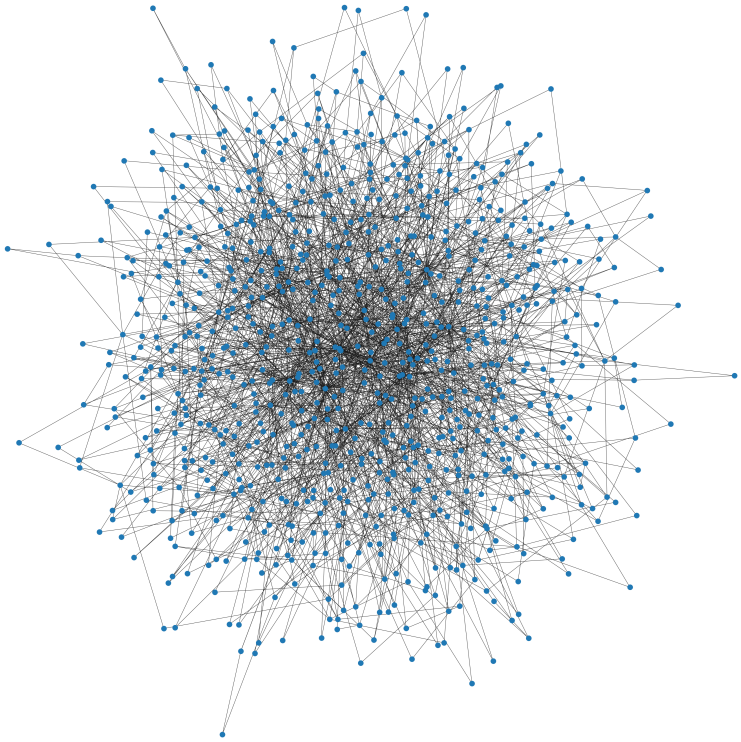
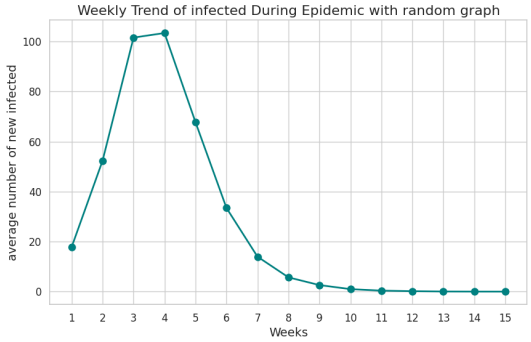
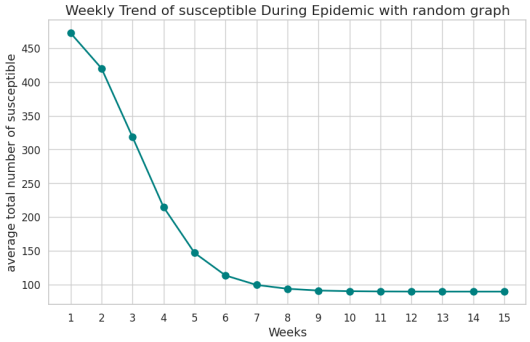


Figure 3

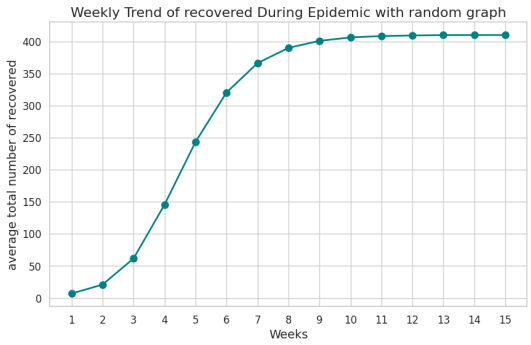
For problem 2 again, we simulate the disease spreading in this new setup with 500 people and track it over multiple experiments. This time, each person has, on average, six friends. We compare the results with what we saw in the first network. you see the result in the figure 4 and figure 5



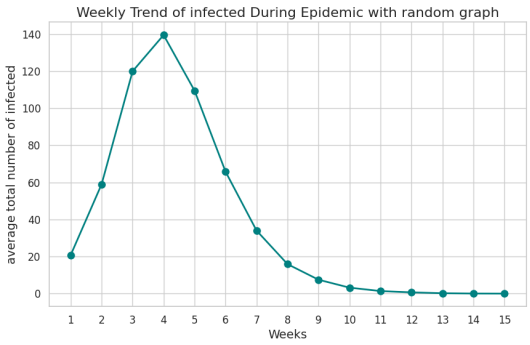
(a)



(b)



(c)



(d)

Figure 4

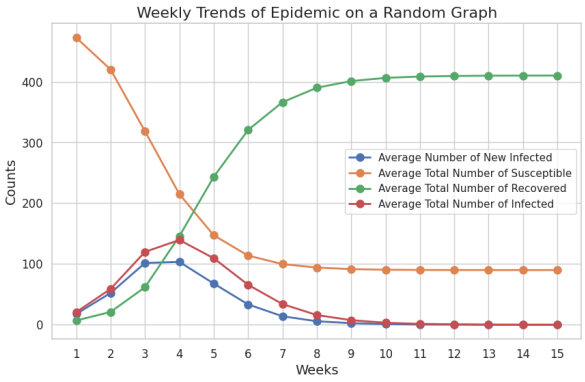


Figure 5

1.3 Simulate a pandemic with vaccination

In this section, we simulate the effects of a vaccination program during an epidemic over a 15-week period in a population of 500 people. The objective of the simulation is to understand how immunizations impact the spread of an infectious disease within a network of people. We have defined a schedule for weekly vaccinations, where a specific number of individuals are vaccinated each week. This schedule is designed to mimic a realistic vaccination rollout, with varying vaccination numbers from week to week. Initially, a small number of people randomly contract the infection. As the simulation progresses, the disease spreads through the population according to predetermined transmission rates and recovery probabilities. The vaccination program intervenes by gradually reducing the population of susceptible individuals.

The population is represented as a graph, with each individual as a node and connections between

them as edges. This graph is generated using a preferential attachment algorithm, a common method for modeling social networks. In such a network, some individuals have more connections than others, resembling the "small-world" networks observed in real-life social interactions.

The core of the script is the `simulate_epidemic_optimized` function, which iteratively performs the following steps for each week:

1. Vaccinate a set number of individuals from the pool of those who are not yet vaccinated.
2. For each individual, determine whether they become infected based on their connections to infected individuals and the disease's transmission rate.
3. Update the status of infected individuals to recovered based on the recovery probability.

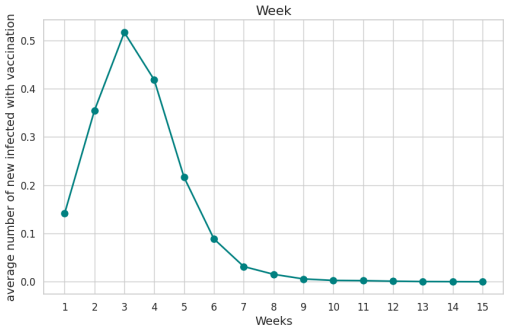
Each individual's condition is monitored using an array with the following values:

- 0: Susceptible (not infected, not vaccinated)
- 1: Infected
- 2: Recovered (and immune)
- 3: Vaccinated (and immune)

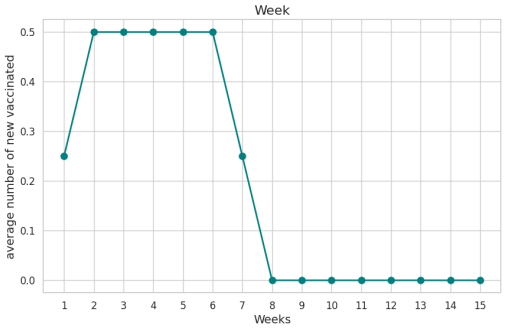
After each weekly cycle, the script collects data on the number of new infections, vaccinations, and the current state of all individuals. By running the simulation 100 times, it calculates the average number of individuals in each state per week. This provides a statistically reliable representation of the epidemic's progression.

The script concludes by visualizing the averaged results using a series of plots. Each plot represents a different aspect of the epidemic, such as the average number of new infections, the number of individuals vaccinated, and the counts of susceptible, infected, recovered, and vaccinated individuals over time.

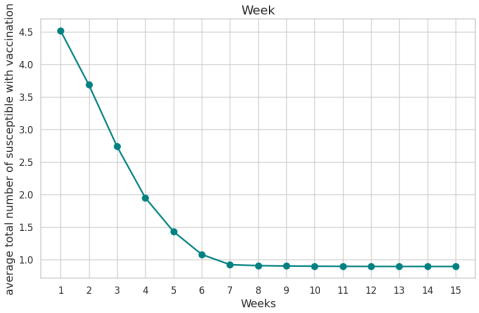
Through these visualizations, the script offers insights into the effectiveness of the vaccination program, illustrating how it can reduce the spread of the disease and protect the population.


Week
average number of new infected with vaccination
Weeks

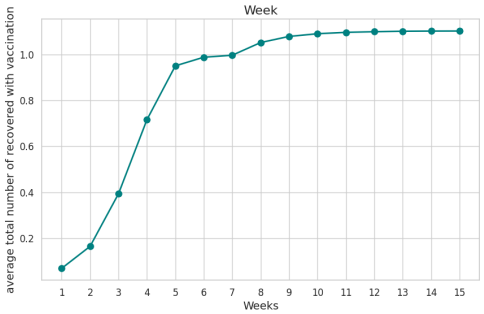
(a)


Week
average number of new vaccinated
Weeks

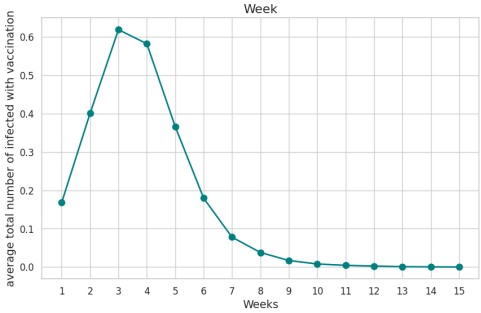
(b)


Week
average total number of susceptible with vaccination
Weeks

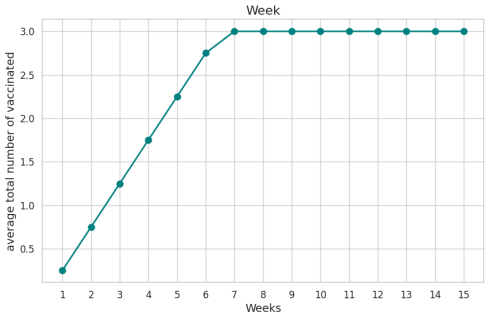
(c)


Week
average total number of recovered with vaccination
Weeks

(d)


Week
average total number of infected with vaccination
Weeks

(e)


Week
average total number of vaccinated
Weeks

(f)

1.4 The H1N1 Pandemic in Sweden 2009

This simulation aims to evaluate the evolution of infections in the Swedish population, taking into account vaccination rates and the spread of the disease over time.

- Vaccination Rate ($Vacc(t)$): The percentage of the population vaccinated each week is represented as $[5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60, 60]$.
- Initial Infections ($I_0(t)$): The estimated number of new infections per week is $[1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0, 0]$.
- Starting parameters for the network model: $k_0 = 10$, $\beta_0 = 0.3$, and $\rho_0 = 0.6$. These parameters represent network connectivity, transmission probability, and recovery rate, respectively.

A gradient-based search algorithm is employed to estimate the optimal parameters (β, ρ, and k) that best approximate the values of newly infected individuals. The parameters are varied within the range:

- $k \in \{k_0 - \Delta k, k_0, k_0 + \Delta k\}$
- $\beta \in \{\beta_0 - \Delta\beta, \beta_0, \beta_0 + \Delta\beta\}$
- $\rho \in \{\rho_0 - \Delta\rho, \rho_0, \rho_0 + \Delta\rho\}$

Through a grid search, 27 combinations are explored, and the set that minimizes the Root Mean Square Error (RMSE) is selected. RMSE is computed as:

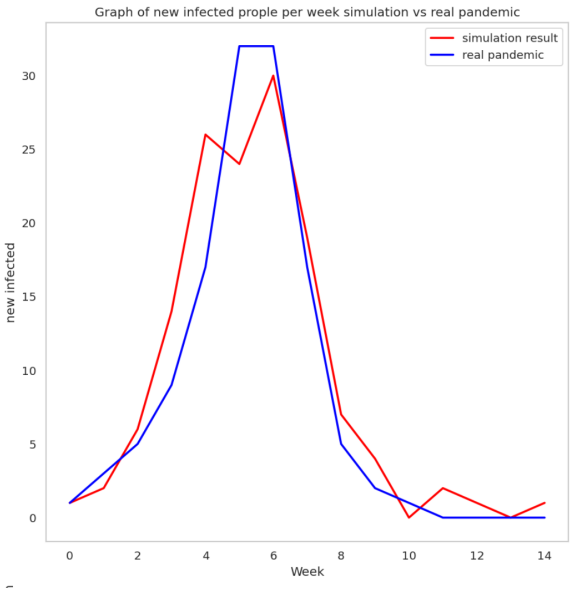
$$RMSE = \sqrt{\frac{1}{15}\sum_{t=1}^{15}(I(t) - I_0(t))^2}$$

where $I(t)$ represents the average number of newly infected individuals each week after 10 iterations. If the RMSE does not improve in an iteration, the delta values are halved, and the search continues around the parameters that yielded the lowest error. The algorithm stops when the same set of parameters is selected as the best in two consecutive iterations.

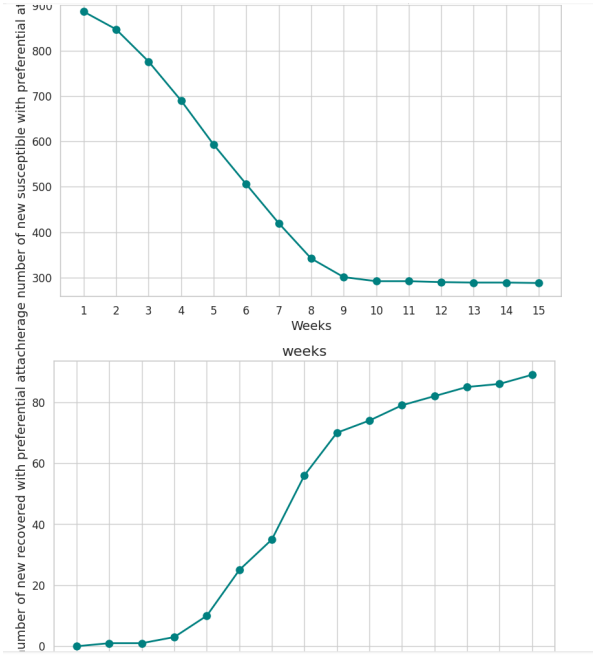
The best parameters found by the algorithm are:

- $\beta = 0.2$
- $\rho = 0.6$
- $k = 9$

A graphical representation of the difference between I_0 and $I(t)$ is provided for a clear comparison of the real and simulated infection data.



(a) Graph of new infected people per week simulation vs real pandemic



(b) Number of new susceptible and recovered with preferential attachment

Figure 7: Comparative data on infection and recovery rates

2 Coloring

Introduction

Graph coloring is a classical problem in graph theory with practical applications in distributed learning algorithms. The objective is to assign a color to each node in an undirected graph such that no adjacent nodes share the same color. This report provides a comprehensive account of a simulation study conducted on a linear graph with 10 nodes and a network of 100 routers. The purpose is to showcase distributed learning methods in potential games, specifically for graph coloring.

2.1 Linear Graph with 10 Nodes

Problem Statement

The study begins with a linear graph comprising 10 nodes. Each node is capable of exhibiting one of two colors: red or green. At the outset, all nodes are initialized to red. At each discrete time step t, a randomly selected node alters its color based on the following probability distribution:

$$P(X_i(t+1) = a | X(t), I(t) = i) = \frac{e^{-\eta(t) \sum_j W_{ij} c(a, X_j(t))}}{\sum_{s \in C} e^{-\eta(t) \sum_j W_{ij} c(s, X_j(t))}} \tag{3}$$

where W_{ij} is the weight between nodes i and j, $\eta(t)$ is the time-decreasing parameter defined as $\eta(t) = \frac{t}{100}$, and $c(s, X_j(t))$ is the cost function given by:

$$c(s, X_j(t)) = \begin{cases} 1 & \text{if } X_j(t) = s \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Implementation

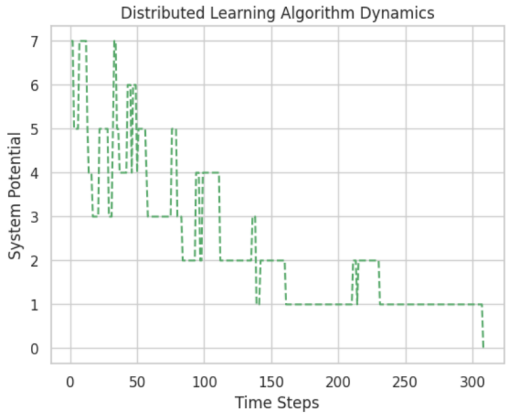
The implementation involves a Python class, `GraphSimulation`, which encapsulates the simulation's logic. Key functionalities include initializing the graph, updating node colors, calculating probabilities and potentials, and visualizing the graph and potential over time.

Results and Discussion

The simulation proceeds with all nodes starting as red. Nodes switch colors based on the defined probability distribution, influenced by neighboring colors and the decay factor $\eta(t)$. The system's potential—a measure quantifying conflict within the graph—decreases over time as nodes settle into a stable configuration where adjacent nodes are differently colored. The simulation concludes when the potential reaches zero, denoting an absence of adjacent nodes sharing the same color.

(a) Initial Graph Configuration

(b) Final Graph Configuration



(c) System Potential

2.2 Network of 100 Routers

Problem Statement

The second part of the study scales up the problem to a network of 100 routers. Each router must be assigned one of eight colors representing different Wi-Fi channels, while minimizing interference. The cost function for this part is:

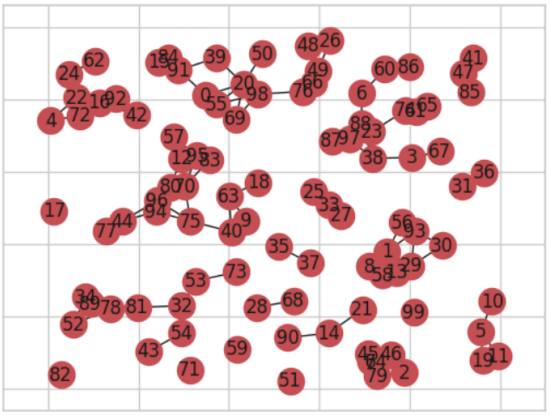
$$c(s, X_j(t)) = \begin{cases} 2 & \text{if } X_j(t) = s \\ 1 & \text{if } |X_j(t) - s| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Implementation

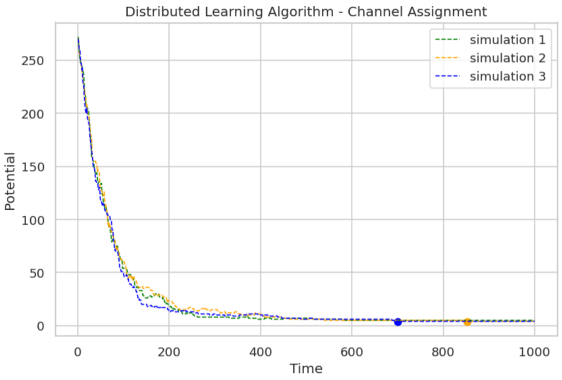
For this scenario, a `ChannelAssignmentSimulation` class was created. It extends the principles of the previous simulation to accommodate more colors and a more complex cost function. The simulation involves adjusting the colors (channels) of routers to achieve a near-zero potential, which indicates minimal interference.

Results and Discussion

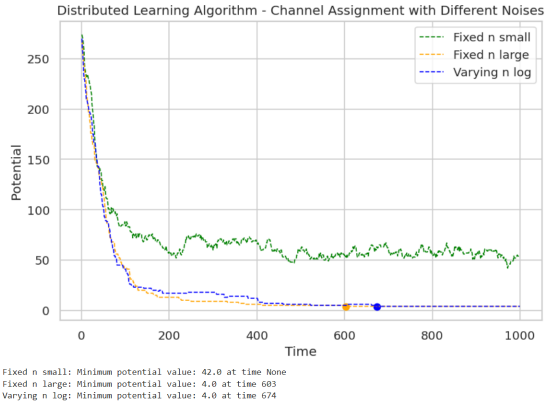
The simulation explores how different learning rates influence the convergence to a solution with minimal potential. The potential function decreases over time as the algorithm iteratively adjusts router channels to reduce interference. The outcome of this simulation informs the effectiveness of distributed learning in managing complex networks, such as Wi-Fi channels in a dense router environment.



(a) Graph



(b) Potential



(c) Potential

Conclusions

The conducted simulations elucidate the distributed learning algorithms' performance in graph coloring. The linear graph model demonstrates how a distributed approach can effectively find a non-conflicting node arrangement. In contrast, the router network