Network Dynamics and Learning

Homework 3

Alessandro Ferì (s305949) Mattia Sabato (s305849)
Marco Tasca (s285174) Riccardo Moroni (s320002)

January 21, 2024

Abstract

The following report is the result of the collaborative work between Alessandro Ferì, Mattia Sabato, Marco Tasca and Riccardo Moroni. When convenient, some of the numerical results have been omitted from this document and reported only in the accompanying Notebooks.

1 Influenza H1N1 2009 Pandemic in Sweden

During the fall of 2009 there was a large pandemic of the H1N1-virus, commonly known as the swine-flu. During this pandemic it is estimated that about 1.5 million people in Sweden were infected. As an attempt to stop the pandemic and reduce excess mortality the government issued a vaccination program beginning in week 40 of 2009. During the weeks that followed they vaccinated more than 60In this homework, you will simulate the pandemic with the goal of learning the network- structure characteristics and disease-dynamics parameters of the pandemic in Sweden 2009. This task will be divided into 4 parts where the focus of each part is to:

1. (a). simulate a pandemic on a known graph; (b) generate random graphs;
2. simulate disease propagation on a random graph without vaccination;
3. simulate disease propagation on a random graph with vaccination;
4. estimate the network-structure characteristics and disease-dynamics parameters for the pandemic in Sweden during the fall of 2009.

1 Exercise 1

1.1.a Simulation of a pandemic on a known graph

We aim to simulate a pandemic on a symmetric k-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = 500$ and a constant node degree of $k = 4$. This pandemic simulation employs a discrete-time,

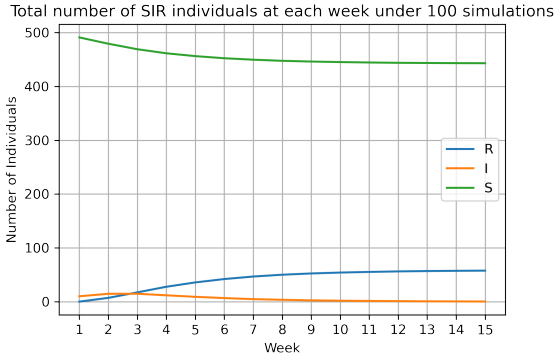
simplified version of the SIR (Susceptible, Infected, Recovered) epidemic model. At a given time t, each node $i \in \mathcal{V}$ can be in one of three states: $X_i(t) \in \{S, I, R\}$, representing Susceptible, Infected, and Recovered, respectively.

The probabilities of transitioning from Susceptible to Infected, and from Infected to Recovered, are defined as follows:

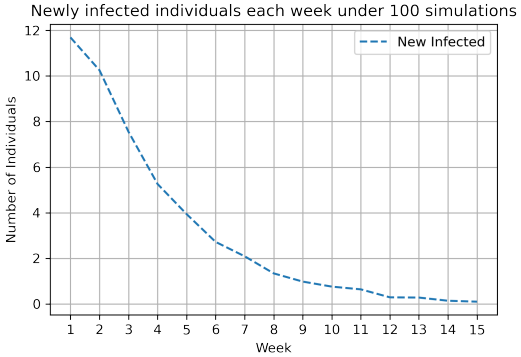
$$P(X_i(t+1) = I \mid X_i(t) = S, \sum_{j \in \mathcal{V}} W_{ij} \delta^I_{X_j(t)} = m) = 1 - (1-\beta)^m$$

$$P(X_i(t+1) = R \mid X_i(t) = I) = \rho$$

where $\sum_{j \in \mathcal{V}} W_{ij} \delta^I_{X_j(t)}$ denotes the number of infected neighbors of node i.
The model will be simulated over a period of 15 weeks, with each week representing one unit of time. We set the infection probability β at 0.3 and the recovery probability ρ at 0.7. Initially, 10 individuals are infected; these individuals are distributed randomly across the graph. Later in this report, we show other three alternative kinds of placements of initially infected individuals. The results of this simulation are shown in Figure 1.



(a) SIR state in each week on average



(b) Average of newly infected

Figure 1

1.1.b Generate random graphs

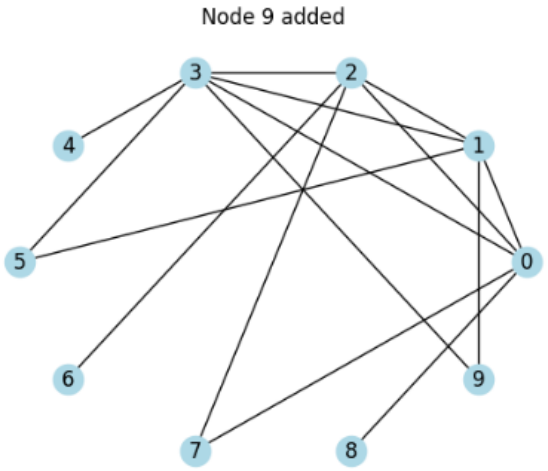
Since we do not expect a symmetric k-regular graph to properly model the society on which we are going to simulate a pandemic, our goal is now to generate a random graph and we do by it following the preferential attachment model. The generation of such a graphs needs just 2 parameters, the number of nodes and the average degree k we want the nodes to have. The algorithm that generates it starts from a complete graph of k+1 nodes and adds one node at a time. When adding a node, it adds $k/2$ undirected links between the new node and other already existing nodes chosen based on some probability that is proportional to their current degrees. So, formally speaking:

$$P(W_{n_t,i}(t) = W_{i,n_t}(t) = 1 |\ \mathcal{G}_{t-1} = (\mathcal{V}_{t-1}, \mathcal{E}_{t-1})) = \frac{\omega_i(t-1)}{\sum_{j \in \mathcal{V}_{t-1}} \omega_j(t-1)}$$

Where n_t is the node being added, $i \in \mathcal{V}_{t-1}$ is any other already existing node, $W(t)$ is the adjacency matrix at the next time-step t and $\omega_i(t-1)$ is the degree of node i prior to adding n_t. The existence of multiple links between the same nodes was avoided, while the scenario in which k is odd was managed by alternatively adding $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$ based on n_t being an odd or even node.
By doing so we were able to generate random graphs with 900 nodes and an average degree equal to k in few seconds.
Here is a little demonstration of what happens at the last step of the creation of a random graph of $n = 10$ nodes following the preferential attachment rule with average degree $k = 3$.

```
i: 9
probs: [0.19230769230769232,
        0.15384615384615385,
        0.19230769230769232,
        0.19230769230769232,
        0.038461538461538464,
        0.07692307692307693,
        0.038461538461538464,
        0.07692307692307693,
        0.038461538461538464]
chosen_nodes: array([3, 1])
```

(a)



(b)

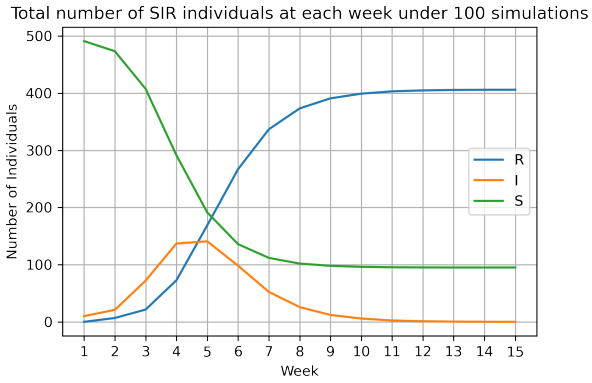
Figure 2

The algorithm started from the complete graph composed of nodes $\{0, 1, 2, 3\}$ and as we can see is now adding the 10^{th} node to the graph, which is labeled as 9 since it starts from 0. The average degree $k = 3$ we are willing to obtain is odd so, being 9 odd, the algorithm adds $\lceil k/2 \rceil = 2$ undirected links between node 9 and the other already existing nodes chosen based on that probability distribution obtained as previously discussed. The resulting graph has 29 undirected links so an average node degree of 2.9 .

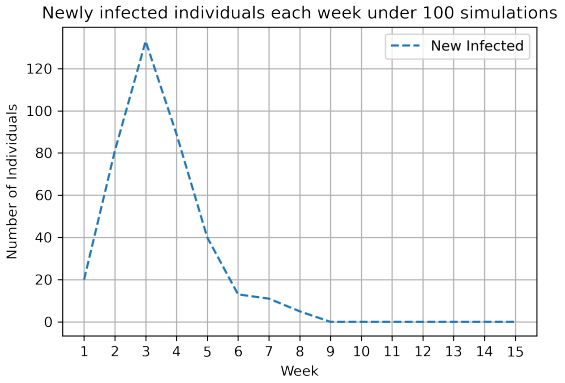
1.2 Simulate disease propagation without vaccination

In this section, we simulate a pandemic on the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which was generated in the previous section using the preferential attachment method. The graph consists of $|\mathcal{V}| = 500$ nodes, with an average degree $k = 6$. The parameters for the interaction and recovery probabilities are set at $\beta = 0.3$ and $\rho = 0.7$, respectively. Figure 3 presents the average system behaviour over 100 simulations.

The initial number of infected nodes is set to 10, and they are randomly chosen. We also considered three alternative kinds of placements of initially infected individuals: *Cluster distribution*, where a random set of closely located nodes are chosen as the infected ones; *Isolated distribution*, where the infected nodes are randomly selected to be far apart from



(a) SIR state in each week on average



(b) Average of newly infected

Figure 3

each other; and *Centrality distribution*, where individuals with the highest degrees are chosen. We simulated the pandemic for $N = 100$ iterations under the same conditions (as shown in Figure 4).

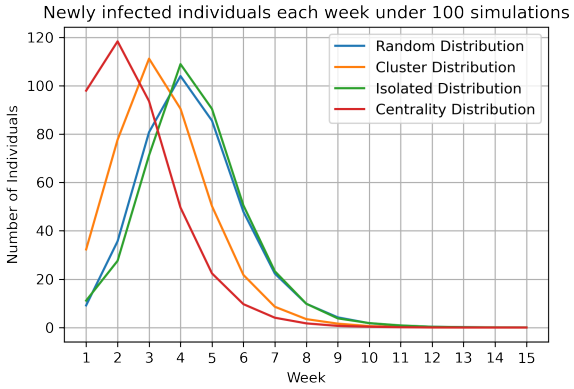


Figure 4: Simulations with different initial distributions of the infected individuals

As observed, the Centrality distribution has a negative impact on the number of newly infected individuals, which can be attributed to the higher degrees of the selected population. The Isolated distribution yields results similar to the Random distribution but progresses slightly slower. Meanwhile, the Cluster distribution negatively impacts the adjacent neighbors, leading to a larger growth compared to the Random distribution.

1.3 Simulate a pandemic with vaccination

Now, we attempt to slow down the pandemic by introducing vaccination. Each week, a fraction of the population is vaccinated according to the schedule:

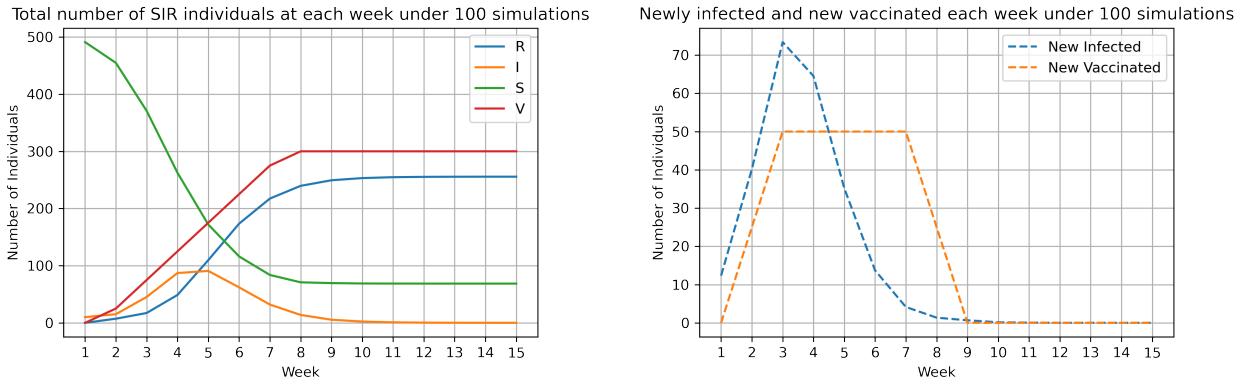
$$\text{Vacc}(t) = [0, 5, 15, 25, 35, 45, 55, 60, 60, 60, 60, 60, 60, 60, 60], \tag{1}$$

where Vacc$[t]$ represents the percentage of the population already vaccinated at the beginning of week t while Vacc$[t+1]-$Vacc$[t]$ represents the percentage of population being vaccinated during week t . Vaccinated nodes cannot spread the disease to others, but we implemented it without the assumption that as soon as an infected receives the vaccine it becomes just vaccinated, it will be counted as vaccinated and will not spread the disease to others but is still going to be also counted as infected and will become recovered according to ρ as before. Moreover, since the choice of the nodes that receive the vaccine is performed among the nodes that didn't receive the vaccine yet, some already recovered individuals can be chosen to be vaccinated, in order to avoid seeing the number of recovered individuals diminishing over time, those individuals will be counted twice in the plots below, and that's why S,I,R,V don't sum to 500 but sum to more.

As in the previous simulations, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a random graph obtained with the preferential attachment rule, it has $|\mathcal{V}| = 500$ and an average node degree of $k = 6$. The parameters $\beta = 0.3$ and $\rho = 0.7$ are retained. Prior to the simulation of each week, we compute the number of people that are going to be vaccinated during the week, #Vacc, which also represents the number of newly vaccinated individuals:

$$\#\text{Vacc}(t) = \frac{|\mathcal{V}|}{100} \times (\text{Vacc}(t+1) - \text{Vacc}(t)), \tag{2}$$

with $t \in \{0, \ldots, \#weeks - 1\}$.



(a) SIRV state in each week on average

(b) Average of newly infected

Figure 5: Impact of vaccination on pandemic dynamics

After simulating the pandemic for $N = 100$ iterations, we observe from Figure 5(a) that the system reaches a trapping configuration after 10 weeks, as the number of infected individuals decreases to zero and the states of the population remain unchanged. As expected, the number of newly infected individuals significantly decreased compared to the previous simulation, due to the introduction of the vaccine (Figure 6).

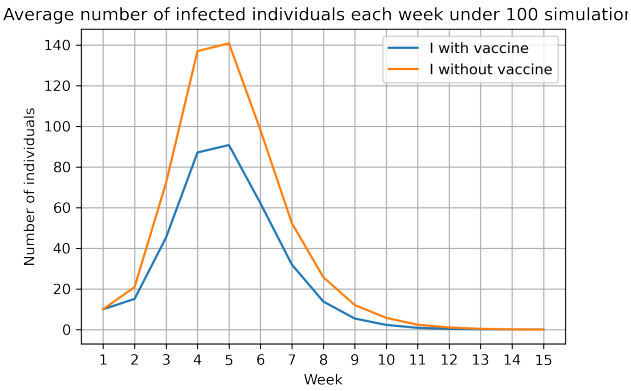


Figure 6: Comparison between Infected individual w/o vaccine

1.4 The H1N1 pandemic in Sweden 2009

In this section, we estimate the social structure of the Swedish population and the disease-spread parameters during the H1N1 pandemic. To minimize the time spent running simulations, we reduce the population of Sweden by a factor of 10^4. Consequently, the population size during the simulation is represented as $n = |\mathcal{V}| = 934$. This adjustment allows for more efficient simulations while preserving the essential characteristics of the scenario.

We simulate the pandemic between week 42 of 2009 and week 5 of 2010. During these weeks, the fraction of the population that received vaccination was:

$$Vacc(t) = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60, 60]. \tag{3}$$

For the scaled version, the number of newly infected individuals each week was:

$$I_0(t) = [1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0, 0]. \tag{4}$$

We used the following algorithm to find the set of parameters (β, ρ, k) that best matches the real pandemic:

Algorithm: Start with an initial guess of the parameters, $k_0 = 10$, $\beta_0 = 0.3$, and $\rho_0 = 0.6$, along with increments $\Delta k = 1$, $\Delta\beta = 0.1$, and $\Delta\rho = 0.1$.

1. For each set of parameters (k, β, ρ) in the parameter-space $k \in \{k_0 - \Delta k, k_0, k_0 + \Delta k\}$, $\beta \in \{\beta_0 - \Delta\beta, \beta_0, \beta_0 + \Delta\beta\}$, $\rho \in \{\rho_0 - \Delta\rho, \rho_0, \rho_0 + \Delta\rho\}$:

   (a) Generate a random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ using the preferential attachment model developed in Section 1. The average degree should be k, and there should be $|\mathcal{V}| = 934$ nodes in the graph.

   (b) Starting from week 42, simulate the pandemic for 15 weeks on $\mathcal{G}$. Use the method developed in Section 1 with the vaccination scheme described above. Do this $N = 10$ times, and compute the average number of newly infected individuals each week, $I(t)$.

(c) Compute the root-mean-square error (RMSE) between the simulation and the real pandemic data:

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

where $I(t)$ is the average number of newly infected individuals each week in the simulation and $I_0(t)$ is the true value of newly infected individuals each week.

2. Update k_0, β_0, and ρ_0 to the set of parameters yielding the lowest RMSE. If the result was the same set of parameters, the algorithm should stop.

After executing the algorithm, we found $k = 9$, $\beta = 0.21$, $\rho = 0.56$ with a $RMSE = 4.94$. The results are shown in Figure 7 and Figure 8

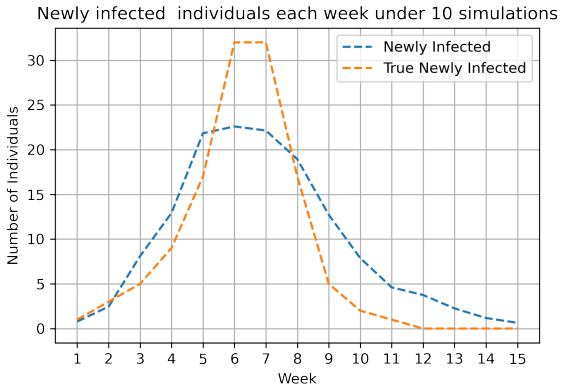


Figure 7: Comparison between I_0 and I

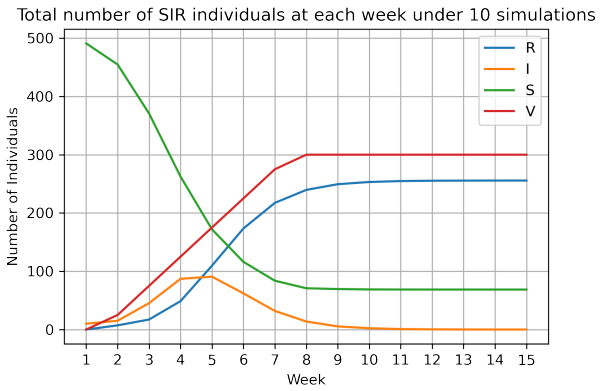


Figure 8: SIRV state in each week on average

1.5 Challenge

The gradient-descent based algorithm presented in the previous section is characterized by many aspects that prevent it from reaching optimal solutions. At every step, it needs to always evaluate a cube of 2^3 combinations of parameters, which drastically increases to 3^3 if some of the parameters are also allowed to stay put. Moreover, the choice of the values of the steps for each of them is crucial and there is not a unique and clear way to decrease them over time. In order to explore alternative ways to perform such an optimization, we have implemented an Evolutionary Strategy (ES) of type (μ, λ), denoted as a *comma strategy*, where, at each generation, only the top μ individuals from the λ generated are retained, while the remaining are discarded. After having tried several hyper-parameter configurations, we have been able to reduce more than half the previous lowest RMSE using $\mu = 3$, $\lambda = 12$ and 20 generations. In order to understand how the algorithm works, it is first necessary to introduce a bit of notation.

Let $T \in \mathbb{N}^+$ be the maximum number of generations over which letting evolve the population, and let $t = 1, \dots, T$ be a generic instant in time, where we reserve the special value $t = 0$ for the first iteration. Let $\phi_i^{(t)} = (k_i^{(t)}, \beta_i^{(t)}, \rho_i^{(t)}), \phi_i^{(t)} \in \Phi$, with $i = 1, \dots \lambda$ and $t = 1, \dots, T$ be a generic individual i at generation t, i.e. a vector of parameters, and $\mathcal{L}(\phi_i^{(t)}) : \Phi \to \mathbb{R}$ be a loss function associated with a single individual, i.e. the RMSE.

We can define, for each individual, a mutation $\Xi_i^{(t)} = (\xi_{i,k}^{(t)}, \xi_{i,\beta}^{(t)}, \xi_{i,\rho}^{(t)}), \Xi_i^{(t)} \in \mathcal{X}^{(t)}$, where the components are random variables. In particular, we define $\xi_{i,k}^{(0)}$ to be distributed as $P(\xi_{i,k}^{(0)} = x) = \frac{1}{4}, x \in \{-2, -1, +1, +2\}$, $\xi_{i,\beta}^{(0)} \sim \mathcal{N}(0, 0.1)$ and $\xi_{i,\rho}^{(0)} \sim \mathcal{N}(0, 0.1)$, for $t = 0$, whereas, for $t > 0$, $\xi_{i,k}^{(t)}$ is such that $P(\xi_{i,k}^{(t)} = x) = \frac{1}{2}, x \in \{-1, +1\}$, $\xi_{i,\beta}^{(t)} \sim \mathcal{N}(0, \sigma_\beta^{(t)})$ and $\xi_{i,\rho}^{(t)} \sim \mathcal{N}(0, \sigma_\rho^{(t)})$. Even though very similar, the first formulation is used only at the first iteration to induce a bigger exploration through the solution space, while the second is used in later steps to exploit what has already been found.

Let's now define a mutation function $f(\phi_i^{(t)}, \Xi_i^{(t)}) : \Phi \times \mathcal{X}^{(t)} \to \Phi$ that, received as input an individual and a vector of mutations, simply returns the mutated individual, adopting two different behaviours depending on t and defined as follows

$$f(\phi_i^{(t)}, \Xi_i^{(t)}) = (k_i^{(t)}, \beta_i^{(t)}, \rho_i^{(t)}) + \mathbb{1}_{t=0}(\xi_{i,k}^{(t)}, \xi_{i,\beta}^{(t)}, \xi_{i,\rho}^{(t)}) + (1 - \mathbb{1}_{t=0})\delta^{(j)}(\xi_{i,k}^{(t)}, \xi_{i,\beta}^{(t)}, \xi_{i,\rho}^{(t)})$$

where j is a value randomly chosen from $\{1, 2, 3\}$ and $\delta^{(j)}$ is a vector composed of all zeros but in position j where it is 1. Finally, we can define the two sets $\Gamma^{(t)} = \{\gamma_0^{(t)}, \dots, \gamma_\mu^{(t)}\}$, to be the population at time t, i.e. the individuals that will play the role of parents for the next generation, and $\mathcal{O}^{(t)} = \{\phi_0^{(t)}, \dots, \phi_\lambda^{(t)}\}$, to be the offsprings generated at time t. When $t = 0$, i.e. only the starting configuration is available and $\mathcal{P}^{(0)} = \{(10, 0.3, 0.6)\}$, we generate λ offsprings by mutating $\phi_0^{(0)}$ through $f(\phi_0^{(0)}, \Xi_0^{(0)})$, obtaining as a result the set $\mathcal{O}^{(0)} = \{f(\phi_0^{(0)}, \Xi_0^{(0)})_1, \dots, f(\phi_0^{(0)}, \Xi_0^{(0)})_\lambda\}$. By definition, a mutation needs to induce only small perturbations on an individual; however, only for the first step, when generating the initial offsprings, we mutate all the components of the vector. This first disruptive mutation is done in order to generate individuals as much different as possible from the starting point in order to better explore the space of the solutions. Once generated, we compute the loss

$\mathcal{L}(\phi_i^{(0)}), i = 1, \ldots, \lambda$ for all the offsprings belonging to $\mathcal{O}^{(0)}$, which is a proxy for the fitness of each, and then we keep only the top-μ individuals into $\mathcal{P}^{(1)}$. Finally, we repeat until convergence, until we reach T or until a local minima is reached, in which case it could be useful to stop and restart the algorithm. Notice how, in general, when generating $\mathcal{O}^{(t+1)}$, there is a non-zero probability that the same individual $\phi_i^{(t)} \in \mathcal{P}^{(t)}$ will be mutated more than one time, since being $\mu < \lambda$, there is the need to sample with replacement many times from $\mathcal{P}^{(t)}$. However, this causes no problems, since being involved real numbers in the encoding of the individuals, the probability of having equal individuals quickly converges to 0 as generation passes by. Finally, do notice also how, while being the generation of new individuals characterized by some randomness, the survivors of each generation are chosen deterministically, leaving no room for uncertainty. During the experiments, we have noticed that, many times, two similar individuals shared too different losses/fitness. This is not allowed since one core assumption in ES is that, if $\phi_i \sim \phi_j$, then also $\mathcal{L}(\phi_i) \sim \mathcal{L}(\phi_j)$. In order to get more reliable estimates, and so to obtain a smoother fitness landscape, previously characterized by spikes, we increased the number of simulations from 10 to 30, which has indeed proved to be effective. Do note that, for any generation greater than the first, only one parameter, randomly chosen for each individual, will be mutated; a mutation, by definition, should induce a small perturbation in the solutions space, so varying all the three parameters at once would generate a too drastic change. Moreover, in order to leverage the best solutions found, at each generation the standard deviation of the gaussian random variables characterizing the mutations are decreased in order to decrease the exploration and increase the exploitation. To do this, a linear decay function has been used, so that, at every step, the standard deviation is updated as $\sigma^{(t)} \leftarrow (\sigma^{(0)} - \sigma^{(T)})(1 - \frac{t}{T})^p + \sigma^{(T)}$, with $p = 1$, $\sigma^{(0)} = 0.09$ and $\sigma^{(T)} = 0.005$, for both β and ρ. The difference with respect to the gradient-descent based algorithm is that, through the use of random gaussian perturbations, a much larger range of values can be explored. Moreover, with a continuously decreasing standard deviation, the best parameters are not lost lost through time but instead better explored. Finally, in order to never get lost in the solutions space, at each generation we store the best individual, i.e. the one with the lowest RMSE. If in the next generation there is no individual with a lowest value, the best individual is reintroduced in the population, otherwise it is substituted by the new one. This also mimics the behaviour that characterizes *plus strategies* of the form $(\mu + \lambda)$.

k	β	ρ	RMSE
10	0.2152	0.792	2.864
9	0.2656	0.887	2.576
11	0.1621	0.646	2.301

Table 1: ES Results

Thanks to this technique, we have been able to strictly outperform the gradient-descent based method, lowering significantly the value of the loss several times and reaching configurations as the ones shown in Table 1. The behaviour of the RMSE associated with the

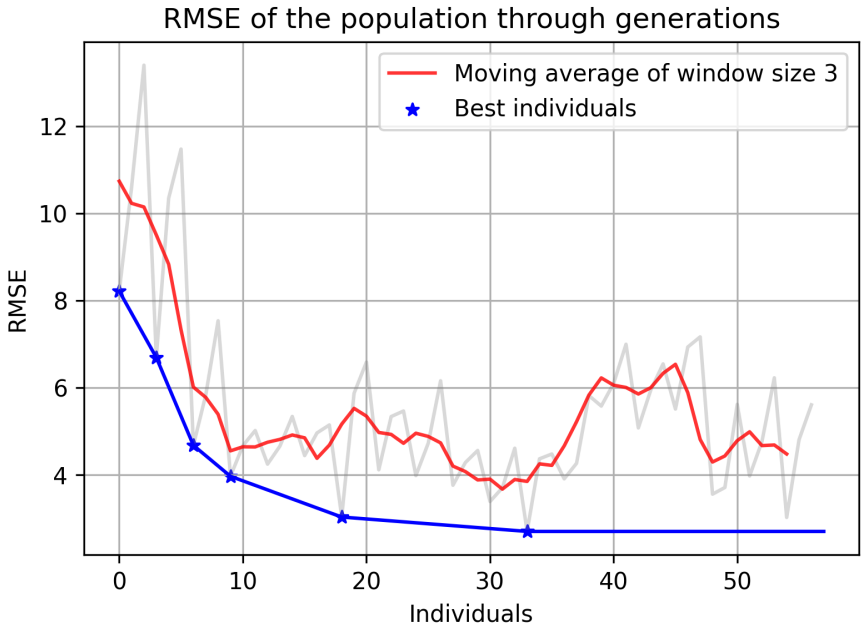


Figure 9: Evolution of the RMSE

individuals in the population is instead shown in Figure 9, where both a moving average and the best individuals through time are also highlighted. It has to be noted, however, that being this method strongly stochastic, it may need several initialization in order to find new solutions and do not get stuck in local minima. A way to solve this would be to promote diversity in the solutions through the use of different *islands*, in which evolving different populations that communicate with each other with a certain small rate. As another future improvement, it can be considered to make use of methods that update the standard deviations of the gaussians through state of the art covariance based approaches since, in the current implementation, the two are independent one from each other, even though this could be not completely true.

2 Coloring

In this part, we will study graph coloring as an application of distributed learning in potential games. The aim of graph coloring is to assign a color to each node in a given undirected graph, such that none of the neighbors of a node have the same color as that node. We will begin with a simple line graph to illustrate the distributed learning algorithm, and then look at a more general example, which can be seen as a distributed solution approach to assign non-interfering channels to wifi access points.

2.a Line graph, with 10 nodes

In this section, we study the noisy best response dynamics over a line graph with 10 nodes, and two possible colors. Denote the i-th node state by $X_i(t)$ and the set of possible states by $C = \{red, green\}$. At initialization, each node is red, i.e, $X_i(t) = red$ for all $i = 1, ..., 10$. This initial node state can be seen in Figure 10a.

Every discrete time instance t, one node $I(t)$, chosen uniformly at random, wakes up and updates its color. Consider the simple graph $G = (\mathcal{V}, \mathcal{E}, W)$, the potential game $(\mathcal{V}, \{\mathcal{A}_\rangle\}, \{u_i\})$ and the node state function $X_i(t)$. Suppose:

$$\mathcal{A}_i = \mathcal{C} = \{\text{red}, \text{green}\} \wedge X_i(0) = \text{red}, \forall i \in \mathcal{V}$$

And the general behaviour:

$$\mathbb{P}(X_i(t+1) = a | X(t) = x, I(t) = i) = \frac{e^{\beta u_i(a, x_{-i})}}{\sum_{a' \in A_i} e^{\beta u_i(a', x_{-i})}} \tag{5}$$

Where:

$$\begin{cases} \beta = \eta(t) = \frac{t}{100} \\ u_i(a, x_{-i}) = -\sum_j W_{ij} c(a, x_j) = -\sum_j W_{ij} \vec{1}_{\{a = x_j\}} \end{cases}$$

With potential function:

$$\phi(t) = \frac{1}{2} \sum_{i,j \in \mathcal{V}} W_{ij} \vec{1}_{\{x_i = x_j\}} \tag{6}$$

Leveraging the above modelization, we can effectively solve the complex NP-complete problem of coloring graphs. In this context, we treat it as a potential game, and we'll try to minimize the potential function $\phi(t)$, using a specific kind of learning dynamics: the NBR (noisy best response) dynamics, defined using the logit dynamics above.

we'll try to reach a $\phi(t) = 0$, which would mean achieving both a Nash equilibrium and a solution to our coloring problem -they are exactly two in a line graph- with no adjacent nodes of the same color.

After the initialization of the line graph, we create and run the function **NBR_dynamics** in Python, for 2000 discrete time steps. This is a simulation that- at each time step- chooses a node at random, and then a color for that node, according to the probability distribution 5 and the two functions $\eta(t)$ and $c(a, X_i(t))$, described above.

The noisy best response dynamics, in this case, is a reversible Markov chain, whose stationary distribution can be explicitly computed. As $\eta \to +\infty$, the dynamics over the action set $\mathcal{A}_i$ converges to a uniform probability on the best-response set $\mathcal{BR}_i(X_{-i}(t))$, so that the noisy best response dynamics reduces to the best response dynamics. The corresponding transition rate matrix is the hypercube Λ so defined:

$$\Lambda_{xy} = \frac{e^{-\eta(t) \sum_j W_{ij} \vec{1}_{\{y_i = x_j\}}}}{\sum_{a \in \mathcal{A}_i} e^{-\eta(t) \sum_j W_{ij} \vec{1}_{\{a = x_j\}}}}$$

with $\Lambda_{xy} = 0$ if $x, y \in \mathcal{X} = \prod_{i \in \mathcal{V}} \mathcal{A}_i$ such that $\sum_{i \in \mathcal{V}} \vec{1}_{\{x_i \neq y_i\}} > 1$: they differ in more than one entry. In the following Figure 10b we can see the resulting configuration of the line graph.

(a) Initial configuration of line graph (b) Final configuration of line graph

Figure 10

In Figure 11a is reported the evolution of $\phi(t)$ and $\eta(t)$ at each discrete time step, while in Figure 11b we reported, for completeness, the evolution of $\phi(t)$ averaged across 100 simulations. We can notice how at first $\phi(t) = 9$, which are the number of adjacent couples of nodes with the same color, and $\eta(t) = 0$. After a while, the potential evolves until it converges to 0, while the noise increases linearly. In the long run $\eta(t) \to +\infty$. The result is perfectly coherent with the theoretical behaviour of such dynamics, which tends to a Nash equilibrium. In our case, the game is potential, and so it can be described and solved effectively by minimizing $\phi(t)$.

The final stable value of $\phi(t) = 0$ allows us to confidently affirm that the graph is 2-colorable, and the Nash equilibrium is also a solution of the coloring problem.

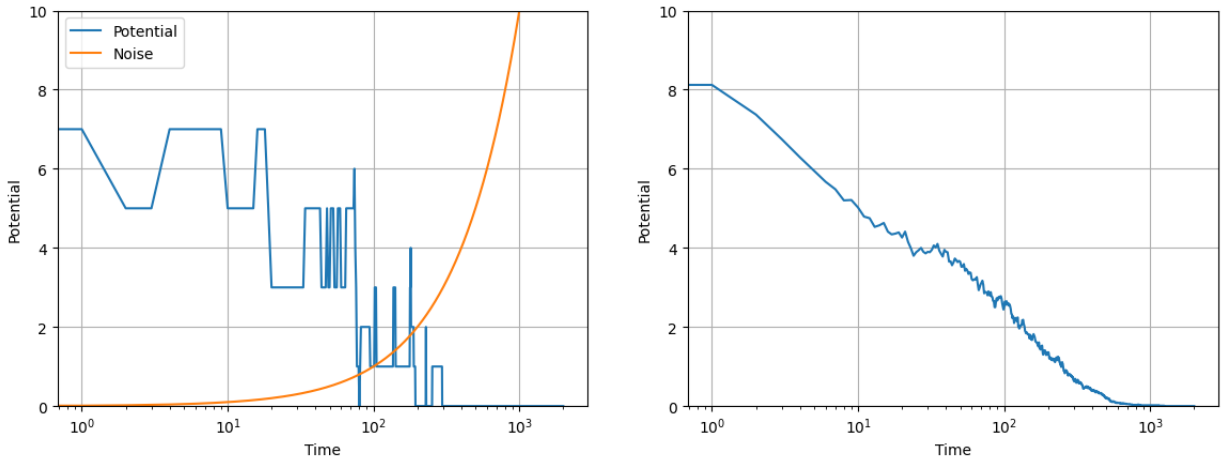
We effectively solved a coloring problem, which is **NP-complete**, by leveraging the above distributed learning dynamics.

Every graph in this exercise is plotted with a logarithmic scale on the x-axis, to emphasise the evolution through discrete time of the different dynamics.

2.b Assignment of wifi-channels to routers

Next, we use the coloring algorithm for the problem of assigning wifi-channels to routers. The adjacency matrix of a network of 100 routers is given in **wifi.mat** and the routers' coordinates are given in **coord.mat**. Here, a link between two nodes means that the two routers are able to interfere with each other. The set of possible states is:

$$C = \{1 : red, 2 : green, 3 : blue, 4 : yellow, 5 : magenta, 6 : cyan, 7 : white, 8 : black\}$$



(a) Potential and noise evolution

(b) Potential averaged on 100 simulations

Figure 11

where colors represent frequency bands, and the cost function is:

$$c(s, X_i(t)) = \begin{cases} 2 \texttt{ if } X_j(t) = s, \\ 1 \texttt{ if } |X_j(t) - s| = 1, \\ 0 \texttt{ otherwise} \end{cases}$$

The cost function $c(s, X_j(t))$ symbolizes that routers that are close by should not use channels with the same frequency band or a frequency band right next to each other.

Finally, we decided to run the experiments with the initial configuration: $X_i(0) = red, \forall i \in \mathcal{V}$, as shown in Figure 12a, and for 2000 discrete time steps, which are enough to let the learning dynamics converge.

We used the same function **NBR_dynamics** described before, with the new defined cost function. Then we plotted the corresponding final configuration, visible in Figure 12b.

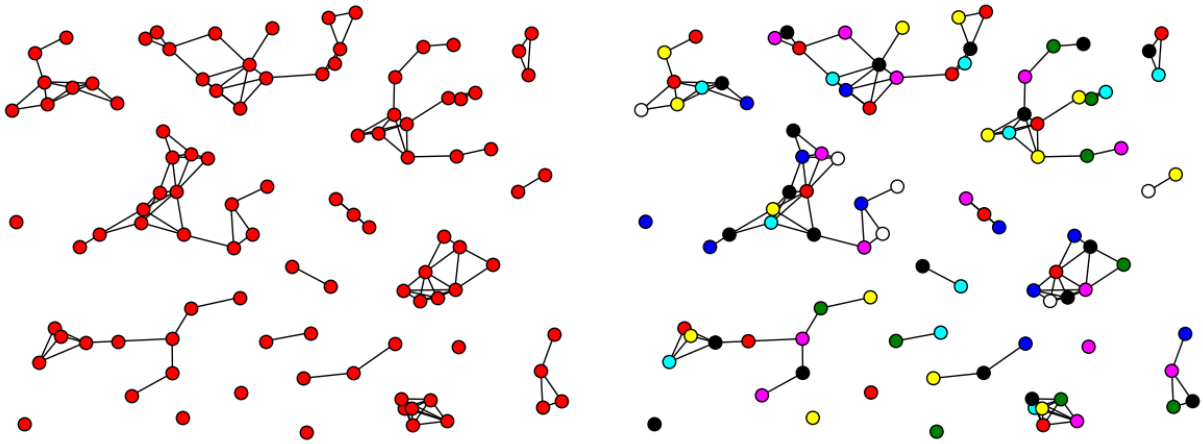
As we can see in Figure 13a, the potential never converges to 0, instead, it happens to stabilize at exactly 4.0, such that:

$$\lim_{t \to +\infty} \phi(t) = 4$$

This means that a perfect solution is never found, and there always be some conflicts between routers. Nevertheless, the final configuration is still a Nash equilibrium, and the noise parameter allows us to affirm that, with high probability, we are not stuck in a local minima. In other words, the solution probably satisfies the following relation:

$$X(t_{final}) \in \operatorname*{argmin}_{x \in \mathcal{X}} \{\phi(t)\}$$

It's interesting to notice how there are different possible configurations at $\phi(t) = 4$, such that each one represents a different Nash equilibrium. We empirically proved it, by running the simulation multiple times.



(a) Initial configuration of routers

(b) One of the final possible routers' configurations

Figure 12

Moreover, to better explore the solution's space, we run the simulation again, with different initial node configurations, as shown in Figure 13b. We noticed how, in all scenarios, 2000 time steps are enough for convergence, and $\phi(t)$ stabilizes at exactly 4, in almost the same time.

It's interesting to notice how, $\forall c \in \mathcal{C}$ such that $X_i(0) = c, \forall i \in \mathcal{V}$, the behaviour of $\phi(t)$ is quite similar, with $\phi(0) > 250$ and $\phi(t_{final}) = 4$. On the opposite, choosing a random initial configuration always corresponds to a $\phi(t) < 100$, but ends up with the same value of 4. This shows how choosing a random initial state decreases the number of initial conflicts among routers, but cannot improve the final solution, neither the time of convergence.

2.c Play with noise $\eta(t)$

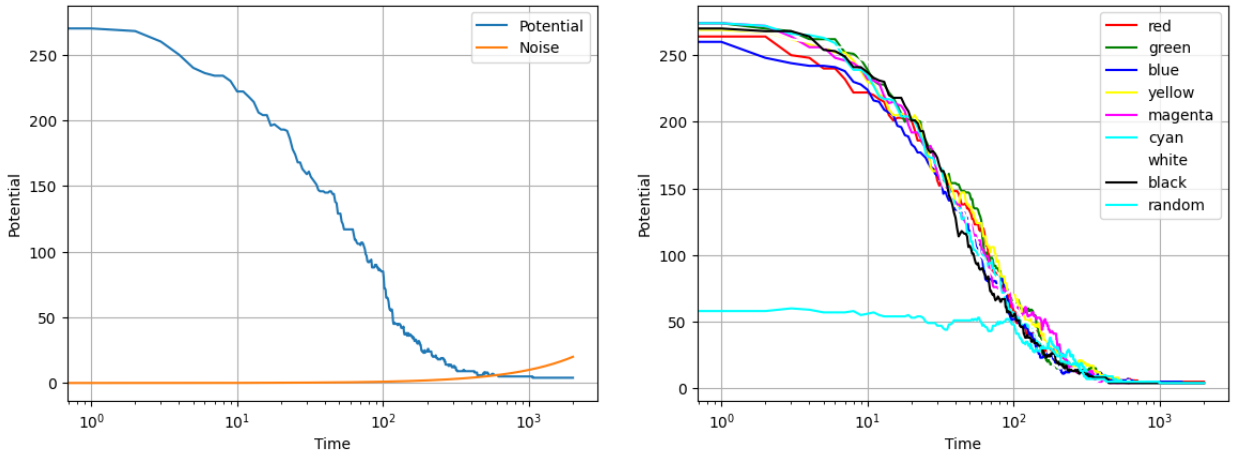
In this section, we evaluated what happens for different choices of $\eta(t)$. In order to being able of experimenting with a high variety of values, we needed a very precise tool, way more precise than **numpy** Python library, after some attempts -better described in the attached Python notebook-, we chose **mpmath** Python library, setting the precision of computations at 100 decimal positions.

We first tried with constant values:

$$\eta \in \{0, 1, 100, 1000\}$$

We believe a quite interesting result is given by setting $\eta = 0$, as a matter of fact as $\eta \to 0$, the dependence on the cost function $c(a, b)$ vanishes, and the dynamics converges to a uniform probability distribution on the action set $\mathcal{A}_i$.

In other words, the dynamics reduces to a random walk on G_Λ such that:



(a) Potential and noise evolution with red initialization

(b) Potentials evolution for different initial node states

Figure 13

$$\Lambda_{xy} = \begin{cases} \frac{1}{|\mathcal{A}_i|} \texttt{ if } \sum_{i \in \mathcal{V}} \vec{1}_{\{x_i \neq y_i\}} = 1 \\ 0 \texttt{ otherwise} \end{cases}$$

On the opposite, we already seen how while $\eta \to +\infty$, the NBR dynamics converges to the corresponding BR dynamics. This can be seen in Figure 14a, where, increasing η we notice how the dynamics converges and stabilizes at the usual $\phi(t)$, while as η decrease, the final value of $\phi(t)$ is more and more unstable.

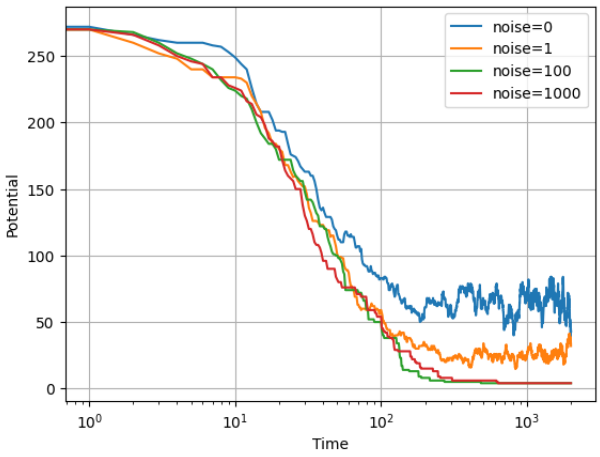
Then we tried with different values of α, modifing the noise function such that:

$$\eta(t) = \frac{t^\alpha}{100}$$

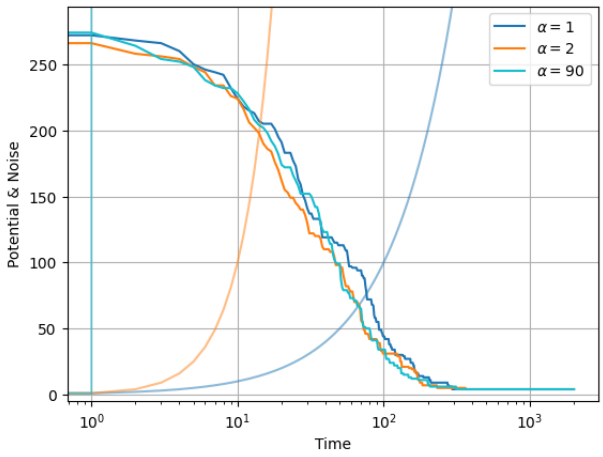
with:

$$\alpha \in \{1, 2, 90\}$$

In this last experiment, we tried to show how, incrementing the speed at which $\eta(t) \to +\infty$, the noisy best response dynamics should converge to a best response dynamics in less and less time, with the risk of getting stuck in a local minima, and never reach the best set of configurations, corresponding to $\phi(t) = 4$. In order to better explore this behaviour, we set α to higher and higher values, pushing the limits of **mpmath**. Nevertheless, even setting $\alpha = 90$ does not result in a bigger final value of the potential. We can conclude that in this scenario, it is very difficult, if not impossible, to find a stable Nash equilibrium with potential higher than $\phi(t)$. it would be very interesting to explore the complete solution space, given by $G_{\mathcal{NBR}}$ with the corresponding $\phi(t)$ for each different state, unfortunately this would be outside the current scope of this homework. In Figure 14b we reported the final result, with the evolution of $\phi(t)$ and, in the same color, the corresponding $\eta(t)$ through discrete time.



(a) Potential with different constant noises



(b) Potentials and noises with different α

Figure 14