Homework 2 - Network Dynamics and Learning

Fabrizio Pisani s305391, Matteo Zulian s310384

December 17, 2023

The code and the report has been done in collaboration without a strict division of work.

Exercise 1

Markov Chain

$$\Lambda = \begin{bmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 1/3 & 0 & 1/3 & 0 \end{bmatrix}$$

In this exercise, we studied a single particle performing a continuous time random walk in the following network.

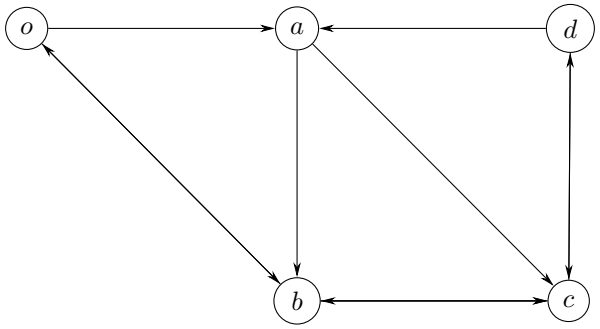


Figure 1: Closed network

Before answering the requests, we computed the normalized weight matrix P, the matrix $\overline{P}$, the vectors w (degree vector) and $\overline{\pi}$ (stationary probability

vector):

$$w = \sum_{j \neq i} \Lambda_{ij} = \begin{bmatrix} 0.6 \\ 1 \\ 0.83333333 \\ 1 \\ 0.66666667 \end{bmatrix} \tag{1}$$

The maximum value w^*, of w, is 1.

$$P = \frac{\Lambda_{ij}}{w_i} = \begin{bmatrix} 0 & 0.66666667 & 0.33333333 & 0 & 0 \\ 0 & 0 & 0.75 & 0.25 & 0 \\ 0.6 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0.33333333 & 0 & 0.66666667 \\ 0 & 0.5 & 0 & 0.5 & 0 \end{bmatrix} \tag{2}$$

$$\overline{P}_{ij} = \frac{\Lambda_{ij}}{w^*}, \quad i \neq j, \qquad \overline{P}_{ii} = 1 - \sum_{j \neq i} \overline{P}_{ij}$$

$$\overline{P} = \begin{bmatrix} 0.4 & 0.4 & 0.2 & 0 & 0 \\ 0 & 0 & 0.75 & 0.25 & 0 \\ 0.5 & 0 & 0.16666667 & 0.33333333 & 0 \\ 0 & 0 & 0.33333333 & 0 & 0.66666667 \\ 0 & 0.33333333 & 0 & 0.33333333 & 0.33333333 \end{bmatrix} \tag{3}$$

$$\overline{\pi} = \overline{P}' \overline{\pi} = \begin{bmatrix} 0.2173913 \\ 0.14906832 \\ 0.26086957 \\ 0.1863354 \\ 0.1863354 \end{bmatrix} \tag{4}$$

Ex 1.A

We implemented simulations of a random walk of a particle, starting from node b. The simulation iterates through the steps of the process. At each step, the particle jumps to a next state based on the matrix P. We recorded the *transition_time*. If the particle returns to the specified state (node b, in our case), the simulation stops and the return time is recorded. We operated several simulation and we computed the average return time.

The value produced by the simulation is in the range [4.4, 5.2].

Ex 1.B

In order to compute the theoretical return-time $E_b[T_b^+]$, we applied the following theorem:

If a graph G is strongly connected, then the expected return time satisfies:

$$E_b[T_b^+] = \frac{1}{w_b \overline{\pi}}.$$

We computed the theoretical return-time applying this formula and we obtained this result: $E_b[T_b^+] = 4.6$. This value is slightly different than the simulation one, but it is quite normal considering that the theoretical return time is an expectation, i.e. an average value.

Ex 1.C

In order to compute the simulation hitting time $E_o[T_d]$, we adapted the code of the exercise 1.*a* to this new case: the starting node and the ending node are not the same anymore. When the particle reaches the destination node d, the *hitting time* is recorded in a vector, and in the end we computed the average hitting time. The value produced by the simulation is in the range [10.5, 11.2].

Ex 1.D

In order to compute the theoretical hitting-time $E_o[T_d]$, we applied the following theorem: if a graph G is strongly connected, then the expected return time satisfies

$$E_o[T_d] = \frac{1}{w_o} + \sum_j P_{oj} E_j[T_d]. \tag{5}$$

To solve (5), we implemented an algorithm that solves this system of equations, knowing that $E_d[T_d] = 0$ because the time to go from d to d is zero :

$$\begin{cases} E_o[T_d] = \dfrac{1}{w_o} + P_{oa}E_a[T_d] + P_{ob}E_b[T_d] \\ E_a[T_d] = \dfrac{1}{w_a} + P_{ab}E_b[T_d] + P_{ac}E_c[T_d] \\ E_b[T_d] = \dfrac{1}{w_b} + P_{bo}E_o[T_d] + P_{bc}E_c[T_d] \\ E_c[T_d] = \dfrac{1}{w_c} + P_{cb}E_b[T_d] + P_{cd}E_d[T_d] \\ E_d[T_d] = 0 \end{cases}$$

Each equation represents the average time for the process to move from one state to another (hitting time). The theoretical hitting time to move from the node o to the node d is: $E_o[T_d] = 10.76$, which is very similar to the hitting time of the simulation.

Opinion dynamics

In this part of the exercise, we interpreted the graph as an opinion dynamics model.

Ex 1.E

We considered the graph in figure 1 and w, P and $\overline{\pi}$ are the same computed in (1), (2) and (4), respectively. In the following, we use the notation π instead $\overline{\pi}$.

Starting from an arbitrary initial condition $x(0)$, we simulated the French-De Groot dynamics on G by computing

$$x(t+1) = Px(t). \tag{6}$$

After one hundred iteration, we obtained, as expected, that we reached consensus, i.e. the entries of the vector $x(100)$ are all the same. This result is guaranteed by the fact that the graph is strongly connected and aperiodic. For this reason, the condensation graph has $s_G = 1$, and this sink component is not periodic.

On the other hand, there is a proposition that states: if a graph G has one sink component, and it is aperiodic, the following result holds

$$\lim_{t\to\infty} x(t) = \alpha \cdot \mathbf{1}, \qquad \alpha = \pi' x(0). \tag{7}$$

In fact, the simulation verified that the consensus value depends on the initial conditions.

Ex 1.F

We considered now a noisy initial state $x(0)$:

$$x_i(0) = \theta + \mu_i \tag{8}$$

where θ is the real value and μ_i are zero-mean random variable representing the noise.

In our example, we considered that the real initial condition is null ($\theta = 0$), so that the actual initial condition is just the noise ($x(0) = \mu$). The expected consensus value is then zero ($\overline{x} = \pi \cdot \mathbf{0} = 0$). We computed in python the noisy consensus value in this way:

$$\overline{x} = \sum_k \pi_k x_k(0) = \theta + \sum_k \pi_k \mu_k = \sum_k \pi_k \mu_k \tag{9}$$

We computed the variance of consensus in this way:

$$\sigma^2_{\overline{x}} = \sum_i \pi_i^2 \sigma_i^2 = 0.2593 \tag{10}$$

which is less than every variance of the single agent σ_i^2. It is call *wisdom of crowds*: in the context of the social learning, the average opinion is less noisy than the opinion of the single agent. However, the simulation did not reach the real value of the consensus, due to the noise.

Ex 1.G

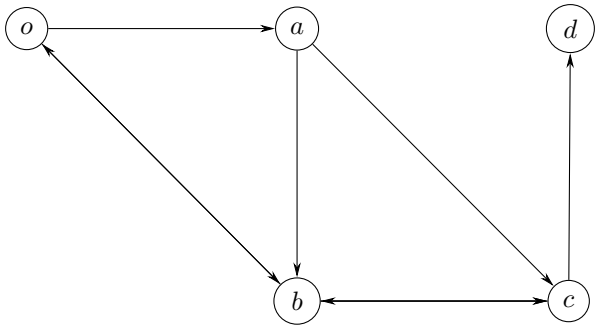


Figure 2: This graph has a sink in the node d

In graph 2 there is a sink component with a single node. In order to study the asymptotic dynamics, we assumed that this problem could be solved as a *stubborn node case*, where the node d is an input node and its "opinion" is constant and not changeable. In order to do that we had to compute the normalized weight matrix P, but matrix W had a row of all zeros, because of the sink node d, so matrix D is non-invertible. We solved this problem by adding a self-loop to node d, so we were able to compute $P = D^{-1}W$ with the new matrix W

$$W = \begin{bmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{(W with zero row)}$$

$$W = \begin{bmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(W with self loop)}$$

From P previously computed we extracted matrices Q and E:

$$P = \begin{bmatrix} Q^{4\times4} & E^{4\times1} \\ F^{1\times4} & G^{1\times1} \end{bmatrix}$$

Therefore, we modeled the distributed linear averaging:

$$\underline{x}(t+1) = Q\underline{x}(t) + Eu.$$

where u is the exogenous input vectors

We solved this problem applying the theory:

$$(I - Q)^{-1} = \sum_{k \geq 0} Q^k$$

and we know that, for every constant input vector u (in this case is a scalar), it holds:

$$\lim_{t \to \infty} \underline{x}(t) = (I - Q)^{-1} E u$$

for every initial state vector $\underline{x}(0)$.

In our case, we defined (as in ex 1.F):

$$x_i(0) = \theta + \mu_i, \tag{11}$$

where $\theta = 0$.

Since graph 2 is not connected, the *wisdom of crowd* does not hold in this case. In fact, there is a single stubborn node, so we expected that the consensus value depended on the value of the input u. The simulation confirmed the theory: we obtained that the consensus value is equal to the input, i.e. the d-th component of noisy initial state vector $x(0)$.

Ex 1.H

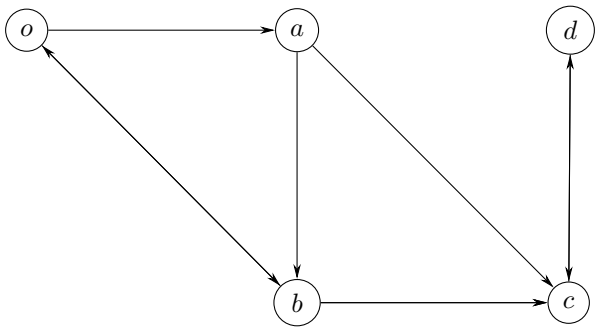


Figure 3: This graph has a sink composed by the nodes d and c

In the graph in figure 3 there is a single sink component, in which there are the nodes c, d. However, this sink is periodic (with period = 2). From the theory we know that for this reason, there is no asymptotic consensus. In fact, running the simulation we observed that on the even iteration we obtained $x_c = x_c(0)$, $x_d = x_d(0)$; on the odd iteration we obtained $x_c = x_d(0)$, $x_d = x_c(0)$.

Using this topology, it is impossible to reach a consensus, for every initial condition.

Exercise 2

Ex 2.A Particles Prospective

The random walk of one hundred independent particles is quite similar to the random walk of one particle, focusing on the single particle. In order to compute the *return time* $E_b[T_b^+]$, we generalized the code written in the ex 1.*a*. In particular, we wrote three nested loops: for each simulation, for each time step, we moved one particle (if that particle has not returned yet). After the movement, we computed the waiting time for the next transition from node i as : $t_{next,i} = \frac{-log(u)}{w_i}$. To do this, we used the rate of the clock of the current position of each particle (Local Clock).

Algorithm 1 Particle Prospective Return Time

```
Initialization Simulation Parameters
for every simulation do
    for every step do
        for every particle do
            Move the Particle
            Waiting Time Computation
            Time Recording
        Return Time Calculation
        Save Minimum Return Time
    Mean Return Time Calculation
```

After running several times the code, the average return time produced by the simulation is in the range [4.4, 5.2]. This result is very similar to the result obtained in the first exercise. This is normal: even if in this exercise there are more particles, they don't influence themselves on their random walk on the graph. Therefore, the average return time is similar to the expected return time.

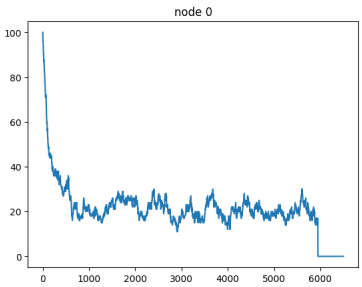
Ex 2.B Nodes prospective

In this case, we focused on the node prospective. The main difference compared to the particle prospective is that, in this case, particles influence the transition time: the more is the number of the particles in a node, the faster is the clock of that node. In this algorithm, we used a Global clock with rate $w^* = \max w_i$. Using this clock means that a particle may stay in the same node after an iteration of the algorithm, that's why we used matrix $\overline{P}$, which has non-zero diagonal, instead of P, for the computation of the objective node of the transition.

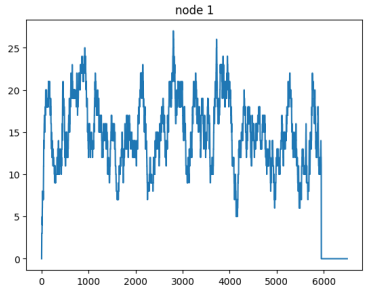
Algorithm 2 Node Prospective

Initialization Simulation Parameters
for *every simulation* **do**
    for *every step* **do**
        Random selection of the current node
        Random selection of the next node
        Move the particle over the nodes
        Update of the particles in the nodes
        compute t_{next}
        Save the simulation results
        Save Minimum Return Time
    Calculation of the mean of the particles in each node

After running the simulation, We obtained this average number of particles in the node: $\left[20.4, 15.6, 28.5, 16.1, 19.4\right]$. In the figure 11a, 11b, 12a, 12b, 13 there is the plot showing the number of particles in each node during the simulation time.

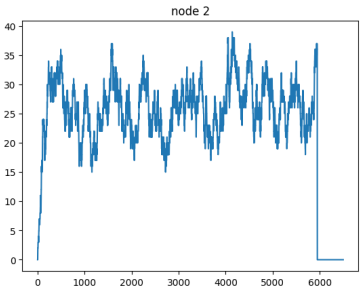


(a) Node o

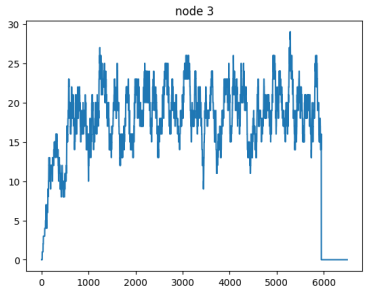


(b) Node a

Figure 4: Plot showing the number of particles in nodes o, a



(a) Node b



(b) Node c

Figure 5: Plot showing the number of particles in nodes b, c

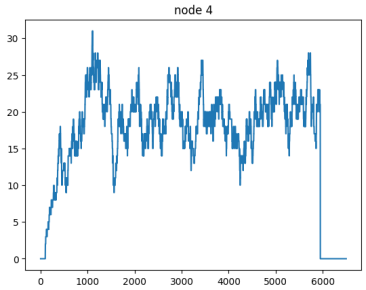


Figure 6: Plot showing the number of particles in nodes d

Comparing the average number of particles in the different nodes at the end of the simulation and the stationary distribution of the continuous-time random walk followed by the single particle, we notice that the second vector can be obtained by multiplying the first one 100 times (approximately), as expected.

Exercise 3

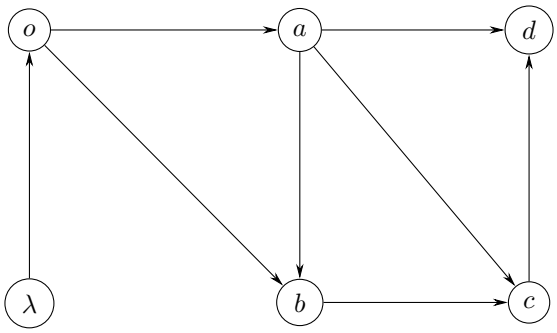


Figure 7: Added node λ connected to o

For the purpose of simulate a network with input rate λ we added a fictitious node to the graph, as in figure 7, that sends particles to node o. The idea is that at each step the algorithm chooses between all nodes the first that is going to have a particle transition and then, based on which node was chosen, it performs several actions to update the network. Every node has a clock (including node λ) that tells the interval of time for the next transition to occur. After moving a particle from the chosen node, all the transition times are computed again. This is possible thanks to the property of Memoryless of Markov Chains: it's not important how much time I've already waited for an event, the time I still have to wait for it to occur has conditional distribution that is a rate-r exponential one. The node d is a *sink* so the matrix W has a row of all *zeros*, but in order to make node d an output of the network we imposed rate $w_d = 2$ so that, every time d clocks, a particle from it is removed.

Ex 3.A Proportional Rate

We simulated the system for 60 time units, with input rate $\lambda = 100$ and proportional rate $r_i = w_i N_i(t)$ for node i at time t. We found out that in the beginning there were only injections of particles in the network from node λ, then when the other nodes had rate big enough, they also started to move particles around. After some time steps the network reaches a sort of equilibrium between input and output. This is because at some point the rates r_o and r_d become the same as λ thanks to $N(t)$ that gradually speeds their clocks until:

$$\lambda = r_o = r_d = w_o N_o(t) = w_d N_d(t). \tag{12}$$

Intuitively, equation (12) can be explained as a "conservation of mass property", in fact everything that comes *in* the network, eventually will come *out* at the same rate. The previous property ensures that even if we increase λ, at some point the network will balance itself provided that it has enough time to do it, as shown in figure 8, 9 and 10.

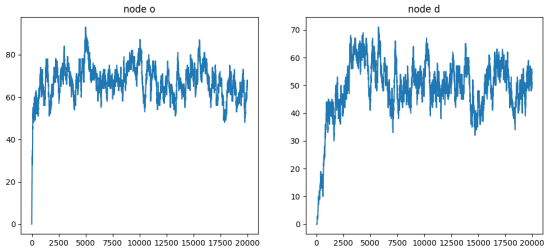


Figure 8: $\lambda = 100$

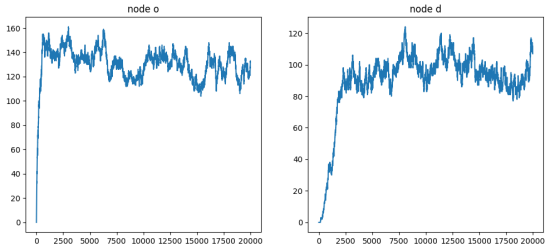


Figure 9: $\lambda = 200$

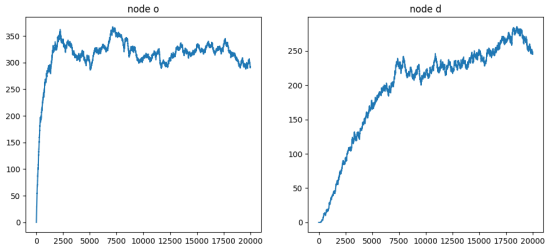
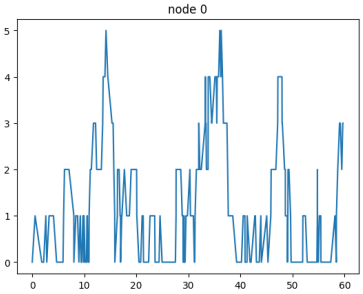


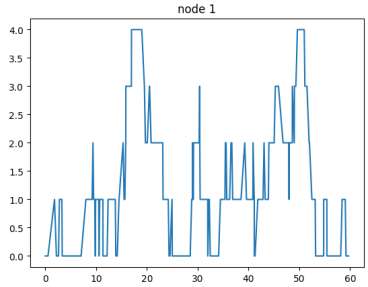
Figure 10: $\lambda = 500$

Ex 3.B Fixed Rate

When we simulated the network using a fixed rate $r_i = w_i$ and $\lambda = 1$ we obtained different results than before. We noticed that the particles were moved slower over nodes, in fact after 60 times unit we had this distribution of particles in nodes:

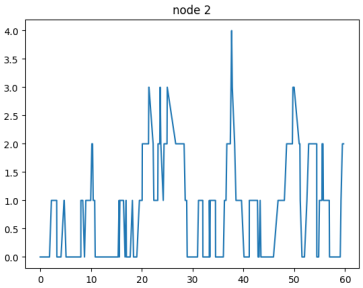


(a) Node o

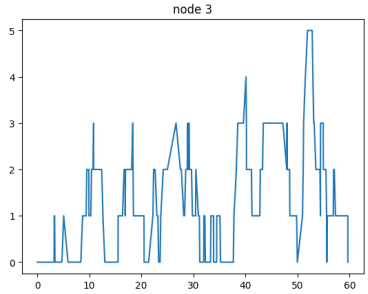


(b) Node a

Figure 11: Plot showing the number of particles in nodes o, a



(a) Node b



(b) Node c

Figure 12: Plot showing the number of particles in nodes b, c

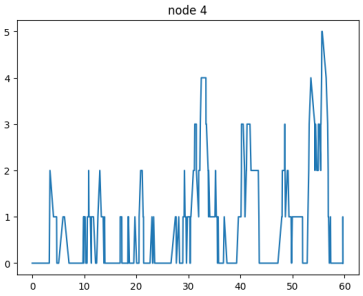


Figure 13: Plot showing the number of particles in nodes d

This kind of Clock doesn't allow the network to adapt to the number of particles, so the equations (12) doesn't apply. As a consequence, when we increased λ, it became the biggest rate of the network ($\lambda \gg w_i$). Thus almost every time step was an injection of particle in node o, leading to a fast increase of particles in that node while the others remained with few particles.

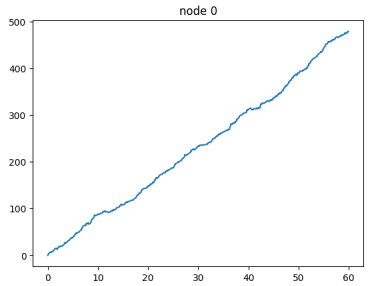


Figure 14: Plot showing node o diverging with $\lambda = 10$

We computed the limit value possible for parameter λ in order to avoid behaviours like in figure (14), as follows.
The idea was to find which was the bottle-neck node of the network, so that we could compute the upper bound of λ. Knowing the network configuration given by the normalized weight matrix P, we were able to compute the probable particles flow rate in each edge, proportional to the input rate λ. It's easy then to see the probable input flow of each node.

For example, node a has one input edge ($o \xrightarrow{p=0.5} a$), with probability 0.5 that the flow coming from node o goes into node a instead of somewhere else. The particle flow coming out of node o is the input rate λ, so for node a we have the following constraint:

$$P_{o,a} \cdot \lambda < w_a, \qquad 0.5\lambda < 1, \qquad \lambda < 2$$

We applied this method to all the nodes of the network and we obtained:

$$\begin{cases} \lambda < w_o & \longrightarrow \lambda < 1.5 \\ 0.5\lambda < w_a & \longrightarrow \lambda < 2 \\ (0.5 + 0.5 \cdot 0.25)\lambda < w_b & \longrightarrow \lambda < 1.6 \\ (0.5 \cdot 1 + 0.5 \cdot 0.25 \cdot 1 + 0.5 \cdot 0.25)\lambda < w_c & \longrightarrow \lambda < 1.33 \\ \lambda < w_d & \longrightarrow \lambda < 2 \end{cases}$$

Clearly node c is the bottle-neck of this network, giving as upperbound value 1.33 for the input rate λ.