Homework 2

Silva Bashllari s299317
Gioele Giachino s295380

December 15, 2024

Disclaimer: We have worked in collaboration for all the exercises by discussing all the points together and reflecting on the possible solutions, by solving them both by hand and through Python libraries. The solutions reflected below are the final versions made with the help of Python Libraries.

1 Exercise 1

The first part of this assignment consists in studying a single particle performing a continuous-time random walk in the network described by the graph in Figure 1 and with the following transition rate matrix:

$$\Lambda = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 1/2 & 0 & 1/3 & 0 \end{pmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} . \tag{1}$$

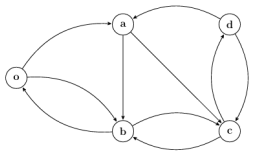

Figure 1: Graph 1

Before we start solving the specific questions, we can note that the graph has the following properties: it is aperiodic and strongly connected. This can also be verified by using the following nx library functions:

```
nx.algorithms.components.is_strongly_connected(G_1)
```

```
nx.is_aperiodic(G_1)
```

1.1

a) What is, according to the simulations, the average time it takes a particle that starts in node a to leave the node and then return to it?
In order to be able to answer this question, we can try to simulate the Continuous Time Markov Chains (CTMC). In order to model the CTMC different approaches might be taken. Here, the chosen approach is using a "global" Poisson Clock with rate *w-star*, thus, the particle has both choices: of jumping to another node or staying in the same node. The way this is done is by taking the already provided transition rate matrix lambda and computing the sum of each row and storing it in a vector w. Then the maximum value of this vector w is taken into account and it is named *w-star* which serves as the rate of the clock. A reminder that the main difference between the discrete and continuous time Markov Chains is that in the latter not only the jumps between the states (nodes) are an element of randomness but also the time elapsed by two consecutive jumps as an exponential random variable with rate r. Then, this clock runs and let's assume the particle is at some node i. When the global clock ticks the particle can either jump at node j with a probability: $Q_{ij} = \frac{\Lambda_{ij}}{\omega_*}$, or stay in node i with a probability: $Q_{ii} = 1 - \sum_{i \neq j} Qij$. Thus, we discretize the continuous time in terms of the ticks of the Poisson Clock. We can observe that the lambda transition rate matrix's rows do not sum up to 1, this is not a stochastic matrix. We do not use that directly to compute the probabilities of jumping to another state/node or staying in the same one but rather we construct the matrix Q, using the function in the code **Qmatrix_Jump_Chain**. The matrix will be the following:

$$\begin{bmatrix} 0.4 & 0.4 & 0.2 & 0 & 0 \\ 0 & 0 & 0.75 & 0.25 & 0 \\ 0.5 & 0 & 0.1667 & 0.3333 & 0 \\ 0 & 0 & 0.3333 & 0 & 0.6667 \\ 0 & 0.5 & 0 & 0.3333 & 0.1667 \end{bmatrix}$$

Then, to simulate the random walk we use a function called **start_simulation**. This function simulates a Markov process with a given transition matrix Q, starting state, and destination state. It computes the sequence of states visited (pos) and the time of each transition (transition_times) until the destination state is reached. The waiting time for each transition is drawn from an exponential distribution, and the next state is selected based on cumulative transition probabilities derived from Q. The simulation ends when the destination state is reached, returning the transition times, visited states, and the total number of transitions.
In order to answer the specific question, on the time it takes a particle to start at node a and then return to it, we run the above simulation function having as input the node a in both initial and destination states, the above provided matrix. After running 100.000 simulations we observe that the average return time is **6.051693085519589**.
Furthermore, we have the first 4 walks and they can be observed in Figure 2. Note how the nodes have been re-labelled from 0-4 corresponding to o,a,b,c,d respectively. Furthermore, as it can be observed in Figure 3, we plotted the scatter plot of the return time value for every 100 simulations, in order not to have a graph which is too polluted. It can be observed that there are outliers, sometimes going all the way to above 25 but for most part the return time values range between 2 and 7.

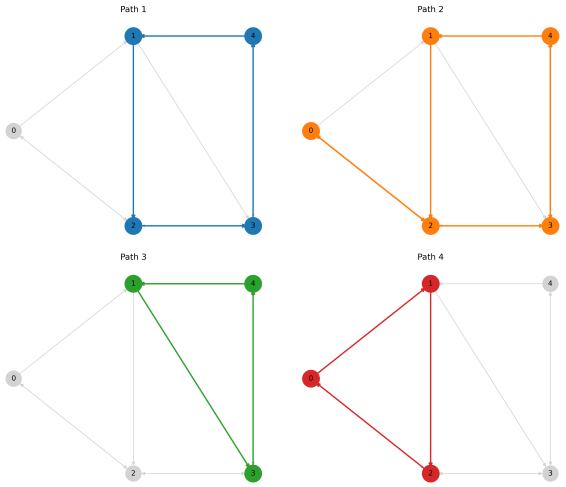

Figure 2: Four different paths the node take.

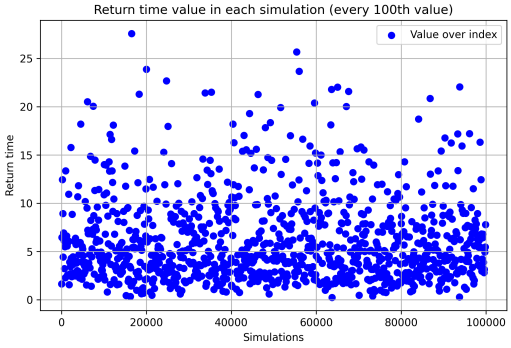

Figure 3: The scatterplot of the value of return time every 100 simulations.

1.2

b) How does the result in a) compare to the theoretical return-time $Ea[T+a]$? (Include a description of how this is computed.)
The theoretical return time is computed as

$$E[T_i^+] = \frac{1}{\omega_i \overline{\pi} i}$$

, where $\overline{\pi}$ is the stationary distribution $\overline{\pi} = P'\pi$. We compute the $\overline{\pi}$ and it has the following entries:**[0.23058252 0.16504854 0.27669903 0.18203883 0.14563107]**. When the above formula is applied to compute the return time from a to a, we obtain the following theoretical expected value: **6.058823529411766**. We then compute the error between the return value we get from the simulations and the expected one, the error in absolute terms is: **0.007130443892177318** and the quadratic error is: **5.084323009948883e-05**.

1.3

c) What is, according to the simulations, the average time it takes to move from node o to node d? Also here we run the same simulation as before but we just alter the start and destination nodes as required. The average time it takes in our simulations to go from o to d is: **10.773257572545207**. Furthermore, we plot every 100th value in the simulations in the scatter plot in Figure 4. We can also observe in this case that there are a lot of outliers but the average value is the one stated above.

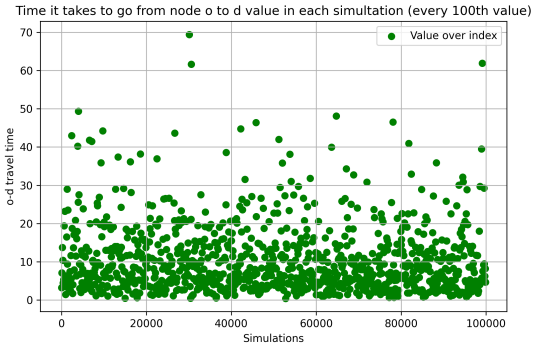

Figure 4: The scatterplot of the value of return time every 100 simulations.

1.4

d) How does the result in c) compare to the theoretical hitting-time $Eo[Td]$? (Describe also how this is computed.) The theoretical hitting-time is computed as:

$$E_o[T_S] = \frac{1}{\omega_o} + \sum_j P_{oj} E_j[T_S]$$

where $S \subseteq V$ and $S = \{d\}$
To obtain $\frac{1}{\omega_o} + \sum_j P_{oj} E_j[T_S]$ we consider $R = V \backslash S$ and $Q = P_{|RxR}, \tau \in R^R$
$\tau = \frac{1}{\omega} + Q\tau \iff (I - Q)\tau = \frac{1}{\omega} \iff \tau = (I - Q)^{-1} \frac{1}{\omega}$.
From our code, the calculations provide the following tau vector: [10.76666667 9. 9.3 4.1]. We take the first element, given that we are interested in hitting node d from node o specifically and that value is **10.76666667**. When computing the error, so the difference between this value and the one we got from the simulations, in absolute terms that is **0.0065909058785411645** and in quadratic terms is **e4.344004029978848e-05**.

1.5

e) Interpret the matrix Lambda as the weight matrix of a graph G = (V, E, Lambda), and simulate the FrenchDeGroot dynamics on G with an arbitrary initial condition x(0). Does the dynamics converge to a consensus state for every initial condition x(0)? Motivate your response.
In French- De Groot Model we interpret the graph as some sort of social network, in which the existence of edges between nodes, for example some edge i-j infers that node i is influenced by j. We simulate the French De Groot Model using the function **sim_french_de_groot** we created in the code that takes as input the number of simulations, the matrix P, the initial vector of opinions and a list of lists we call "opinions" that accumulates how the opinion vectors change as the number of simulations progresses. At each simulation step, the following is done x(t+1)=Px(t). In order to construct the matrix P which is need here, every $P_{i,j} = \Lambda_{i,j}/w_i$.
We run two different experiments with two different initial vectors x01 = [10,20,15,25,40] and x02 = [10,20,15,25,400]. For both cases convergence is reached to these vectors respectively: [20.9565148 20.9564513 20.95669345 20.95621512 20.95675913] and [73.13037485 73.12886604 73.13403501 73.12395856 73.13551051]. The consensus value in the first case is approximately 20.95 and in the second case 73.13. This result is expected given that our graph is both strongly connected and aperiodic and we initially verified. In order to observe the speed of convergence, the two following graphs are constructed as shown in
Figure 5 and Figure 6. Furthermore, we know that as time goes to infinity the consensus value is supposed to converge to $\bar{()} = \pi' x(0), where \pi' is the invariant distribution of the matrix P. We compute \pi' and we find in both cases the following values 20.956521739130434 and 73.$

1.6

f) Assume that the initial state of the dynamics for each node $i \in V$ is given by $x_i(0) = i$, where $i \in V$ are independent random variables with variance

$$\mathbf{Var}(a) = 2,$$
$$\mathbf{Var}(b) = 2,$$
$$\mathbf{Var}(c) = 2,$$
$$\mathbf{Var}(o) = 1,$$
$$\mathbf{Var}(d) = 1.$$

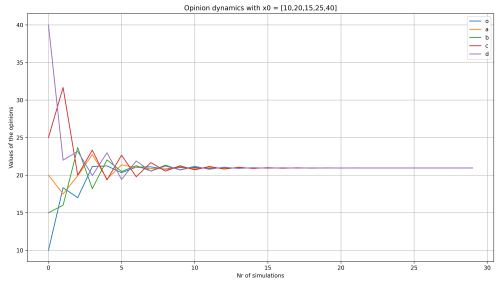

Figure 5: The opinions of the different nodes as the nr of simulations grows.

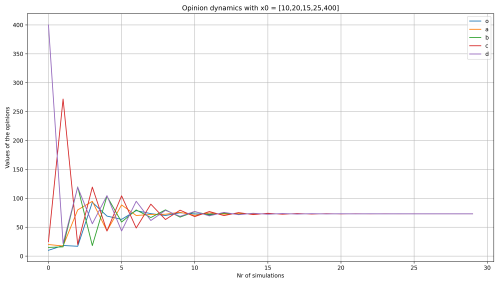

Figure 6: The opinions of the different nodes as the nr of simulations grows.

Compute the variance of the consensus value, and compare your results with the numerical simulations. We compute the variance of the consensus value, which is 0.015543120860464976. This phenomenon is known as Wisdom of Crowds which considers the fact that the crowds are "smarter" and more accurate than individual nodes, given the fact that the variance of the consensus value will be smaller than the variance of single agents.

1.7

g) Remove the edges $(d, a), (d, c), (a, c), (b, c)$. Describe and motivate the asymptotic behaviour of the dynamics. If the dynamics converges to an asymptotic state, how is such a state related to the initial condition $x(0)$?
Removing the above mentioned edges produces a graph, as it is shown in Figure 7. which is neither strongly connected nor aperiodic. We make the alterations in the original transition matrix and compute the new P matrix. We run the French De Groot simulation and we get the evolution of the opinions's vector as shown in Figure 8.

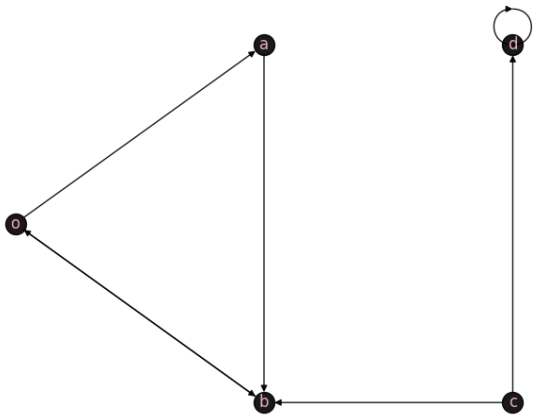

Figure 7: The graph after some edges are removed.

We can observe that for node (d) the opinion value stays always constant at 40. This is expected because (d) is a sink and the direction of the edge c-d infers that d influences c but d is not influenced itself by any other node. Whereas, nodes o-a-b form a component in themselves which is also a sink and they do converge to a consensus value. Last, node c which is both influenced by node d and the trio of nodes a-b-c, comes out with another value. This can be verified also in the condensation graph shown in Figure 9.

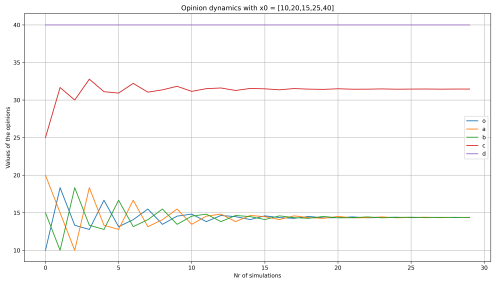

Figure 8: The opinions of the different nodes as the nr of simulations grows.

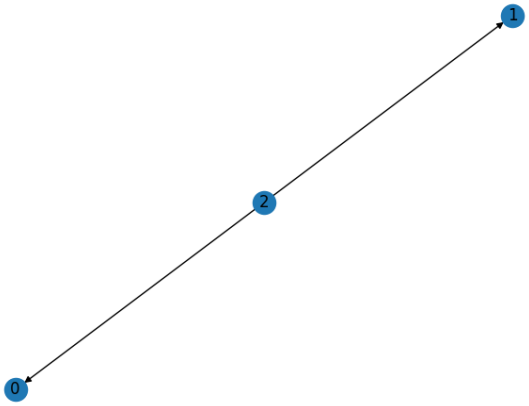

Figure 9: The condensation graph of point (g).

1.8

h) Consider the graph (V, E, Lambda), and remove the edges (b, o) and (d, a). Analyze the FrenchDeGroot dynamics on the new graph. In particular, describe how the asymptotic behaviour of the dynamics varies in terms of the initial condition $x(0)$, and motivate your answer.
We tested the simulation of French- De Groot also in this scenario. The graph after the required edges are removed can be observed in Figure 10 and its condensation graph in Figure 11. After 30 simulations the opinion's vector was the following: [28.33333333 26.66666667 31.66666667 25. 31.66666667]. The evolution of the opinion's vector can be seen in Figure 12 In this instance, this graph can be seen in Figure 10 is not strongly connected but it is aperiodic. It has only 1 sink component that includes the nodes: b,c and d. However, the sink component in itself is not aperiodic, thus the values will oscillate between (K) values, which is the period, and never reach a consensus. Here k is 2 in the sink and we can observe from the opinion vector above that for the last three entries that correspond to the sink there are two values: 31.666 and 25.

2 Exercise 2

2.1

Particle perspective:

- **If N=100 particles all start in node a, what is the average time for a particle to return to node a?**
- **How does this compare to the answer in Problem 1, why?**

We make another simulation function here, similar to the one in the previous exercise but with some minor adjustments to adapt for multiple particles.
The function called **simulation_exercise2** simulates the behavior of 100 particles transitioning between five predefined states based on a Markov chain with transition probabilities provided by the matrix P which is an input. Each particle starts at the state a (state 1), and the simulation continues until all particles return to the initial state. The transition times between states are modeled using a Poisson process, where the waiting time is inversely proportional to a parameter **w_star**. The function outputs the recorded transition times, the states visited by all particles, a tracker for particles that returned to the initial state, and the average transition time per particle.
We make 100 different simulations, we compute the average return time, which corresponds to the value of **6.073742361995771**. We also make a scatter plot to observe how the transition time changes over different simulations, as we can see in Figure 13. We can see that the values oscillate a lot, but within a specific error range from 5.25 to 7.

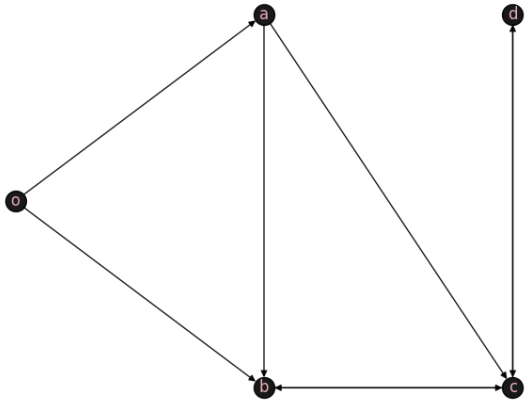

Figure 10: The graph after some edges are removed.

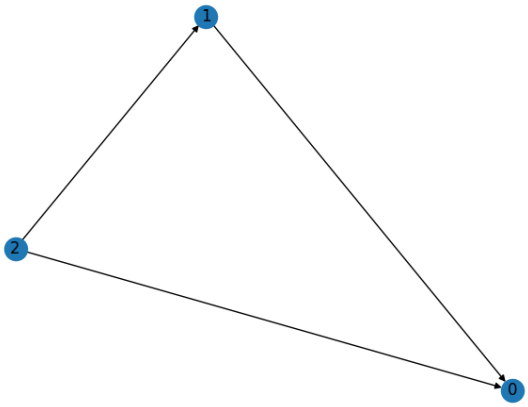

Figure 11: The condensation graph of point (h).

Therefore, we decide to make 1000 simulations and plot an histogram in order to understand the distribution of these values. The histogram can be observed in Figure 14 and it clearly has a normal distribution peaking very close to 6. Thus, we can deduce that this simulation from the viewpoint of particles corresponds in average value in practice to what we already saw in Exercise 1.

2.2

Node perspective:

- **If N = 100 particles start in node a, and the system is simulated for 60 time units, what is the average number of particles in the different nodes at the end of the simulation?**
- **Illustrate the simulation above with a plot showing the number of particles in each node during the simulation time.**
- **Compare the simulation result in the first point above with the stationary distribution of the continuous-time random walk followed by the single particles.**

In this perspective, 100 particles start at node a, and the system is simulated for a cumulative duration of 60 time units. Now we are not interested in the detailed motion of the single particles but only in the number of particles within each node at each fraction of time.
In this simulation in 2.2 we focused on modeling the collective behavior of particles transitioning between nodes in a network, updating cardinalities for all nodes simultaneously and stopping after a predefined simulation time (‘time_units‘). In contrast, the first simulation before in 2.1 tracks the movement of 100 distinct particles individually, with a specific focus on their return to a starting node (’a’) and stopping only when all particles have completed this return.
Running this simulation 100 times, the average number of particles in each node is as follows:
[’o’:23.25, ’a’:15.93, ’b’:27.42, ’c’:18.35, ’d’:15.05]
The distribution of particles over the different nodes as time progresses up to 60 time units can then be observed in Figure 15.
Computing the stationary distribution of the continuous-time random walk followed by the single particles using our created function **comp_eigenvector_one**, we get the following eigenvector:

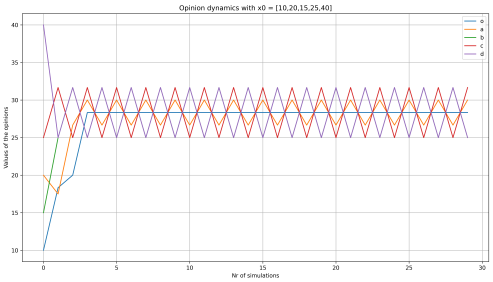

Figure 12: The evolution of the opinion's vector in h.

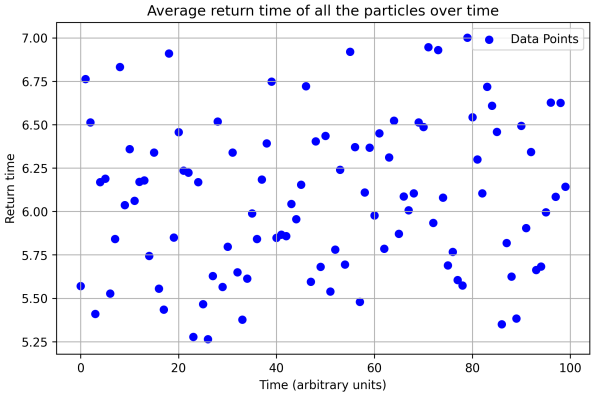

Figure 13: Average return time of all the particles over time

$$[0.23058252 0.16504854 0.27669903 0.18203883 0.14563107]$$

Computing the quadratic error for every node, we can easily observe that the bigger one is the one on node 'b'. All the quadratic error values are the following:

$$[3.67671317e-06, 3.30457545e-05, 6.24514657e-06, 2.13500330e-06, 2.37064992e-05]$$

3 Exercise 3

In this part we consider the open network in 16, with transition rate matrix Lambda_open according to (2).
In this specific system, particles will enter the system at node o according to a Poisson process with input rate λ. Each node will then pass along a particle according to a given rate, similar to what we already did in Exercise 2 in the "node perspective" approach. Let $\omega = 1$ and let $N(t)$ denote the vector of number of particles in each node at time t. We simulate two different scenarios that differ by what rate the nodes will pass along particles: i) proportional rate, and ii) fixed rate. In scenario i), each node i will pass along particles according to a Poisson process with rate equal to the number of particles in the node times the rate of the local Poisson clock of node i, i.e., the node i will pass along particles rate with rate $\omega_i Ni(t)$. In scenario ii), each node i will instead pass along particles with a fixed rate i. Since node d does not have a node to send its particles to, we assume that $d = 2$. When the Poisson clock of node d ticks, you could simply decrease the number of particles in the node by one (if there are any particles in the node). Equivalently think of another node d' connected to node d, such that at every tick of the Poisson clock of d, it sends a particle to node d'.

3.1

Proportional rate:

- **Simulate the system for 60 time units and plot the evolution of the number of particles in each node over time with input rate Lambda = 100**
- **What is the largest input rate that the system can handle without blowing up?**

In this first approach particles pass to the other nodes with a Poisson process with rate **proportional** to the number of particles present in the single node.
t_next is computed as follows:

$$t\_next = -ln(u)/(n\_particles[i] * omega\_i)$$

We consider t_next as infinite if the node has no particles.

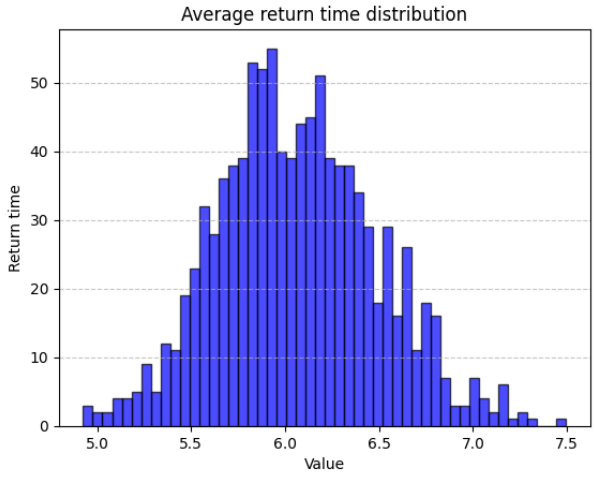

Figure 14: Average return time distribution

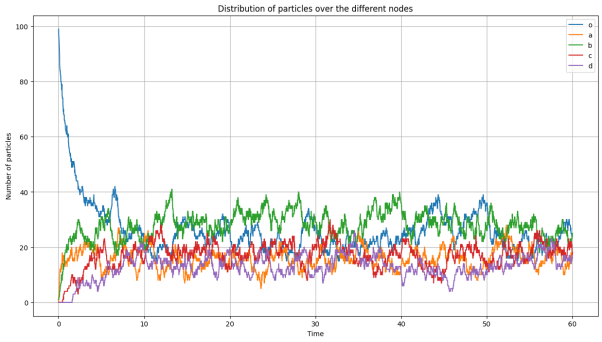

Figure 15: Distribution of particles over the different nodes

In 17 we can observe the system simulation in 60 time units, with an input rate of 100.
To find out the largest input rate for which the system blows up several tests were made with different input rates but with none of these the system exploded, as can be seen for example in 18 where we apply an input rate of 1000. This is due to the fact that nodes adapt their speed according to that of the moving particles.

3.2

Fixed rate:

- **Simulate the system for 60 time units and plot the evolution of number of particles in each node over time with input rate Lambda = 1.**
- **What is the largest input rate that the system can handle without blowing up? Motivate your answer.**

In this second approach the same experiments as before were simulated, the only difference is that in each node i particles will pass along with a fixed rate ω_i.
In 19 we can observe the system simulation in 60 time units, with an input rate of 1. In this case, on contrary with respect to the proportional rate instance, it is sufficient, as we can see in 20, to raise the fixed input rate to 2 to make the system blowing up.

$$\Lambda_{\text{open}} = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 3/4 & 3/4 & 0 & 0 \\ 0 & 0 & 1/4 & 1/4 & 2/4 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \tag{2}$$

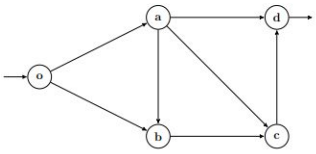

Figure 16: Open network

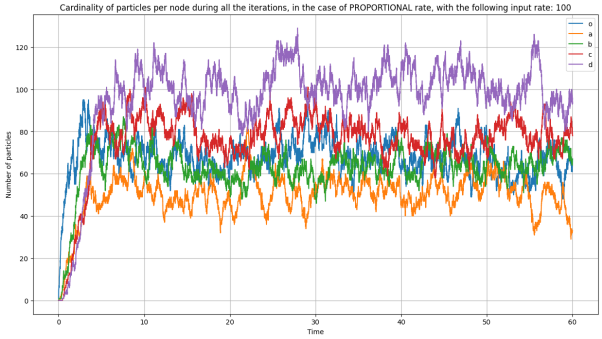

Figure 17: Cardinality of particles per node during all the iterations, in the case of PROPORTIONAL rate, with the following input rate: 100

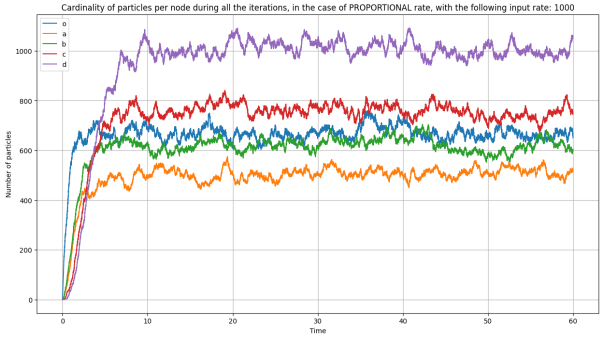

Figure 18: Cardinality of particles per node during all the iterations, in the case of PROPORTIONAL rate, with the following input rate: 1000

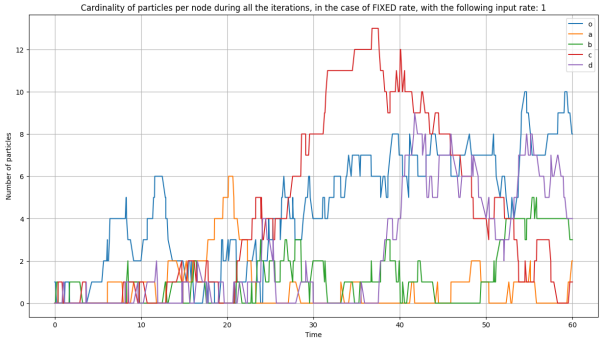

Figure 19: Cardinality of particles per node during all the iterations, in the case of FIXED rate, with the following input rate: 1

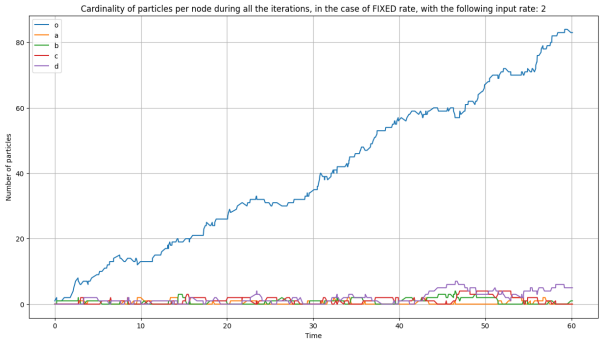

Figure 20: Cardinality of particles per node during all the iterations, in the case of FIXED rate, with the following input rate: 2