FRTN30 Network Dynamics
Hand-in 3

Due: 2024-05-17

Instructions:

- You may implement your solutions in any language you see fit, but the TAs can only guarantee you support with MATLAB/Octave. Your code should be written in a general manner, i.e., if a question is slightly modified, it should only require slight modifications in your code as well. Upload your PDF report together with your code in a separate, runnable file into Canvas. Make sure to name the PDF file with your full name.
- The PDF file should read like a standard lab-report, including a description of what you are doing, a summary of the theory used, proper presentation of results (including readable figures with axis labels) and a short interpretation of the results. Also include your name and stid-id on the first page.
- Comment your code well. Clarity is more important than efficiency.
- Late submission is discouraged: you get 1 points in your exam (out of 25) for each on-time submission.
- Collaboration policy: collaboration such as exchange of ideas among students is encouraged, however, every student has to submit her/his own manuscript (in PDF format) and code, and specify whom she/he has collaborated with and on what particular part of the work.
- Up to five of the best hand-ins may be rewarded with an extra point.
- **Visualization** We have provided you with a code skeleton (for MATLAB) of how to visualize a single particle moving around in a network (in a completely deterministic fashion). This might be useful for you as you go about solving the problems in the hand-in, but *it is in no way needed nor compulsory to solve the hand-in.*

Preparation: In order to be prepared for this hand-in assignment, it is recommended to read through chapter 7 (*Markov chains and random walks*) in the lecture notes, as well as solving the exercises from exercise session 9.

I. Single-particle random walk

Appendix: At the end of this document, you can find the Appendix, which presents the theory for this section in a slightly different way than in the script. However, it should be enough to use the course script.

The first part of this assignment consists of studying a single particle which performs a random walk in the network described by the graph in Fig. 1 and with the following transition rate matrix.

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

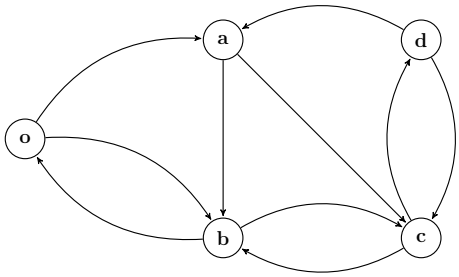

Figure 1: *Closed network in which particles move according to the transition rate matrix* (1).

Your task is to simulate the particle moving around in the network in continuous time according to the transition rate matrix (1). To help you with this, we have provided some hints below. You should use these simulations to answer the following questions:

a) *What is, according to the simulations, the average time it takes a particle that starts in node 'a' to leave that node and then return to it?*

b) *How does the result in a) compare to the theoretical return-time* $\mathbb{E}_a[\bar{T}_a^+]$*? (Include a description of how this is computed.)*

c) *What is, according to the simulations, the average time it takes to move from node 'o' to node 'd'?*

d) *How does the result in c) compare to the theoretical hitting-time $\mathbb{E}_o[\bar{T}_d]$? (Describe also how this is computed.)*

Hint: To simulate a Poisson process with rate r, one can simulate the time between two ticks of the Poisson clock, which we can denote t_{next}. Then, if we draw a random variable u from a uniform distribution, $u \in \mathcal{U}(0,1)$, we can compute t_{next} as,

$$t_{\text{next}} = \frac{-\ln(u)}{r}.$$

It might also be useful to recall that the Poisson process is memoryless:

$$\mathbb{P}(X \geq t+s \mid X \geq t) = \frac{\mathbb{P}(X \geq t+s)}{\mathbb{P}(X \geq t)} = \frac{e^{-r(t+s)}}{e^{-rt}} = e^{-rs} = \mathbb{P}(X \geq s).$$

Furthermore, let's say that you wish to simulate the outcome of when a 6-sided dice is thrown, but where the dice has only 3 numbers $i = 1, 2, 3$, with **probability distribution** $p = (1/6,\ 3/6,\ 2/6)$. The outcome could be simulated by drawing a uniform random variable u in the interval $[0, 1]$. You can then use the **cumulative probability distribution** of u: $\mathbb{F} = (\sum_{i\leq 1} p_i,\ \sum_{i\leq 2} p_i,\ \sum_{i\leq 3} p_i) = (1/6,\ 4/6,\ 6/6)$ and choose the number $i = 1, 2$ or 3 for which $\mathbb{F}_{i-1} < u \leq \mathbb{F}_i$. You can use a similar method but with $p = (P_{i1}, P_{i2}, \ldots, P_{in})$, i.e., row i in the probability matrix P for the discrete-time Markov chain, to randomly select which node the particle should visit next, if the particle currently is in node i.

II. Graph coloring and network games

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.

a) In this example, study a line graph with 10 nodes. Denote the i-th node state by $X_i(t)$ and the set of possible states by $\mathcal{C} = \{\text{red}, \text{green}\}$. At initialization, each node is red, i.e, $X_i(t) = \text{red}$ for all $i = 1, \ldots, 10$. Every time instance t, one node $I(t)$, chosen uniformly at random, wakes up and updates its color.

   The new color (resulting from a node's update), is chosen from a probability distribution given by

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

where the cost is given by

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

In the above expression, $\eta(t)$ is the inverse of the noise. To decide upon a good choice of $\eta(t)$, some heuristics are required, but it is preferable to have $\eta(t)$ increasing in time so that the noise is decreasing. For this exercise you can start with

$$\eta(t) = \frac{t}{100}.$$

To study how close to a solution the learning algorithm is, we consider the potential function, which is given by

$$U(t) = \frac{1}{2} \sum_{i,j \in \mathcal{V}} W_{ij}\, c(X_i(t), X_j(t)),$$

where $\mathcal{V}$ is the set of nodes. If the potential is zero, there are no conflicting nodes and a solution is found.

Your task is to simulate the learning dynamics described above in MATLAB. To animate the dynamics in MATLAB, the functions `gplot` and `scatter` can be useful in order to draw the graph, together with the `pause` function to slow down the animation. Include plots of the potential function in your report and briefly comment on it.

b) 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 $\mathcal{C} = \{1 : \text{red}, 2 : \text{green}, 3 : \text{blue}, 4 : \text{yellow}, 5 : \text{magenta}, 6 : \text{cyan}, 7 : \text{white}, 8 : \text{black}\}$, where colors represent frequency bands, and the cost function is

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

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.

Use $\eta(t) = t/100$ and verify that a near-zero potential solution is found after a sufficient number of iterations. Try with some other function $\eta(t)$ and observe what happens. Include plots of the potential functions for some different tested cases and an illustration of the node coloring corresponding to the smallest potential function (that is, the best obtained solution) in the report. Comment briefly on the obtained results.

Comment on what you observe.

Appendix (for Part I)

A random walk means that the particle's movement is determined by stochastic variables. The properties of these variables can be obtained from the associated **transition rate matrix** Λ. In this matrix, a large element Λ_{ij} means that, if the particle is in state (node) i, it is probable that it will move to state j within a short time. More specifically, defining $\omega_i = \sum_j \Lambda_{ij}$, the particle will in average stay a time $1/\omega_i$ in state i before moving to another state. After this time, the next state j is chosen with probability proportional to the elements on row i in (1). This means that the probability that the next state is j, is given by the element P_{ij} in the normalized weight matrix $P = D^{-1}\Lambda$, where $D = \text{diag}(\omega)$ and $\omega = \Lambda\mathbb{1}$. The fact that the probabilities of the next node only depend on the current node is known as the Markov property, and the process is due to this called a **Markov chain**.

The time S during which the particle stays in a state i before moving to its next state is an exponentially distributed stochastic variable according to

$$\mathbb{P}(S \geq t) = e^{-rt}, \quad t \geq 0, \tag{2}$$

where $r = \omega_i = \sum_j \Lambda_{ij}$ is called the **rate** of the distribution. This means that, in general (unless all nodes have the same out-degree ω_i), the waiting time distributions will be different for the different states. In order to describe the waiting times with exponentially distributed variables of the same rate r, the problem can be reformulated in terms of a **transition probability matrix** $\overline{P}$, with elements

$$\overline{P}_{ij} = \frac{\Lambda_{ij}}{\omega_*}, \quad \omega_* = \max_i \omega_i, \tag{3}$$

$$\overline{P}_{ii} = 1 - \sum_{j \neq i} \overline{P}_{ij}. \tag{4}$$

Together with waiting times given by stochastic variables according to (2), with $r = w_*$, the matrix $\overline{P}$ describes the same system as the matrix (1). This works by choosing the rate $r = w_*$ corresponding to the shortest average waiting time among the nodes, and taking the longer waiting times for the remaining nodes into account by adding self-loops (4) on these so that the particle in average remains equally long as in the original formulation before changing node.

In conclusion, the random walk of the particle can be described by one random process which determines a certain waiting time, and one associated matrix which describes what happens when the waiting time has passed. The waiting-time-determining process uses the sequence

$$T_0 = 0, \quad T_k = \sum_{1 \leq i \leq k} S_i, \quad k = 1, 2, ..., \tag{5}$$

where S_i are exponentially distributed variables, according to (2), with rate $r = w_*$. The sequence (5) is called a **Poisson clock**, and its element T_k is the sum of the first k waiting

times, determined by a sequence of k exponentially distributed variables S_i. The timing-process, which is a so-called **Poisson process** is then given by

$$N_t = \sup\{k \geq 0 : T_k \leq t\}, \quad t \geq 0, \tag{6}$$

where N_t is the number of "ticks" of the Poisson clock (5) at time t, i.e. the number of waiting times that have passed at time t. Each time a waiting time has passed, i.e. each time N_t increases by 1, the particle then moves from the current state i to a state j (which can be itself, in the case of a self-loop) with the probabilities described by (3) and (4).

The state of the particle at time $t \geq 0$ is given by a continuous-time stochastic process $X(t) = U(N_t)$ ($t \in \mathbb{R}_+$), where $U(k)$ ($k = 0, 1, 2, \dots$) is a discrete-time stochastic process given by the Markov chain corresponding to the matrix $\overline{P}$, defined by (3) and (4). $U(k)$ is known as a **jump chain**, since it describes the sequence of moves when the Poisson clock ticks.

The time T_s it takes for the particle to reach a node s is known as the **hitting time**. This is a stochastic variable which depends on which node the particle is in at time $t = 0$. The expectation value of the hitting time of node s when starting at node i is denoted $\mathbb{E}_i(T_s) = \mathbb{E}(T_s \mid X(0) = i)$.

For a *discrete-time* Markov chain, the expected hitting time $\mathbb{E}_i(T_s)$ for node s from node i is given by

$$\mathbb{E}_i(T_s) = 0, \quad \text{if} \quad i = s \tag{7}$$

$$\mathbb{E}_i(T_s) = 1 + \sum_j P_{ij}\mathbb{E}_j(T_s), \quad \text{if} \quad i \neq s. \tag{8}$$

I.e., we have that the hitting time is 0 if the particle is in node s already, or otherwise that we have to wait at least one time unit (for the next step), and then the expected hitting time from the node that is visited next. Thereby, we add to the single time unit the weighted sum of the hitting times from the nodes that could be visited next, weighted by the probabilities that they are chosen in the next step.

In order to use this result for the continuous-time case, we first note that the expected waiting time for a tick of the Poisson clock in the Poisson process (6) is the same all the time, and is given by $\mathbb{E}(S) = 1/r = 1/w_*$, for S in (2). Thus, the expectation value of the time that the jump chain $U(k)$ – corresponding to the matrix $\overline{P}$ – stays in a certain state before changing to the next one (which might be the same) is always the same. Thereby, one can study the hitting time of the jump chain, which is a discrete-time chain, and then multiply the result, i.e. the number of discrete time units, with the expected time between two jumps, in order to get the expected hitting time for the continuous-time process. Note that this means that the normalized probability matrix in (8) should be the transition probability matrix associated with the jump chain $U(k)$, i.e., the matrix $\overline{P}$.

An *alternative* approach for computing the hitting times for the continuous-time Markov chain is to directly use the analogue to the discrete-time equations (7) and (8) for the

continuous-time Markov chain, taking into account the different transition rates in the different nodes, and using the normalized probability matrix P given by the transition rate matrix, i.e., $P = \text{diag}(\omega)^{-1}\Lambda$.

The **return time** T_j^+ for a given node j is the time it takes for a particle to first leave the node and then return back to it, i.e., $T_j^+ = \inf\{t \geq 0 : X(t) = j \text{ and } X(s) \neq j \text{ for some } s \in (0, t)\}$.