Hand-in 3 FRTN30 Gustaf Sundell

gu0147su-s

May 2021

Acknowledgements

When working on this hand-in, I have discussed some of the questions with Fredrik Sidh and Mark Emilson.

1 - Single particle random walk

For the simulation of the random walk, the given transition rate matrix, Λ was used to create a transition probability matrix. As per the instructions, this was done so that the stochastic time for the particle to stay in a node, S, was the same for all nodes. As per the instructions, first the vector $\omega = \Lambda \mathbb{1}$ was calculated. Then, to make the transition probability matrix $\bar{P}$, I followed the instructions in the appendix, see Equation 2.

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

$$\begin{aligned} &\bar{P}_{ij} = \tfrac{\Lambda_{ij}}{\omega_*}, \quad \omega_* = \max_i \omega_i \\ &\bar{P}_{ii} = 1 - \textstyle\sum_{j \neq i} \bar{P}_{ij}. \end{aligned} \tag{2}$$

The resulting transition probability matrix, $\bar{P}$ is shown in Equation 3. In this case, $\omega_* = 1$, which will be significant in task 1.d).

$$\bar{P} = \begin{pmatrix} 2/5 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \end{pmatrix} \tag{3}$$

1.a) Average return time for node a

The simulation was run for 10 000 iterations, which is the number of jumps. Of course, the actual (continuous) time passed for each simulation is different, and is captured in the sequence T_k (see Equation 4), also referred to as the Poisson clock. However, as $\omega_* = 1$, the expectation is that the particle stays in each node for 1 unit of time, and with 10 000 iterations, the actual time taken can be expected to be close to 10 000 time units, which was the case in this script.

$$T_0 = 0, \quad T_k = \sum_{1 \le i \le k} S_i, \quad k = 1, 2, \dots \tag{4}$$

In order to evaluate the average return time for node a, I have utilized the fact that the Markov chain, as well as the Poisson process, are both memory-less. A simple way to calculate the average return time is to start the simulation in node a, and stop it as soon as it has left and then returned, and then re-start the simulation for as many times as one desires. However, the lack of memory of the process comes in handy here, as we can utilize the fact that each time it enters and leaves node a, it behaves just as if it just was initialized in node a. For this reason, I ran the simulation for a large number of iterations (10 000), and calculated the average time that the particle was away from node a (i.e. exited and returned). The times for this were stored in the Poisson clock, and the deltas were averaged. The average return time varied a bit with each simulation, but was around 6.7768. I checked, and it varies both above and below the theoretical value.

1.b) Theoretical return time for node a

The theoretical return time for node a (node 2, as nodes were numerically indexed in the matlab-script) was calculated theoretically by the formula

$$E_2\left[T_2^+\right] = \frac{1}{\omega_2 \bar{\pi}_2} = 6.75 \tag{5}$$

$$L'\bar{\pi} = 0, \quad \mathbb{1}'\bar{\pi} = 1 \tag{6}$$

Which is from Theorem 7.2 in the lecture notes pdf. Note here that $\omega_2 = \omega_* = 1$, which is why no ω shows up in this part of the matlab script. Note also that $L = diag(\omega) - \Lambda$ is the Laplacian matrix of the graph. The result is very close to the one obtained from averaging over the simulation.

1.c) Average hitting time from node o to node d

To calculate the observed average hitting time from node o to node d (nodes 1 and 5 respectively in numerical order) was done similarly to the answer to question 1.a). Utilizing the memory-less nature of the processes, I used the same long simulation and calculated the average time it took for the particle

to reach node 5 every time it left node 1. The result of course also varies across simulations, but varies both above and below the theoretical value. The observed average was 8.9528.

1.d) Theoretical hitting time from node o to node d

Calculating the theoretical hitting time from node 1 to node 5 was done following the same theorem, 7.2 from the pdf. Here, the fact that $\omega_i = \omega_* = 1 \forall \omega_i$ comes in handy once again. Namely, the $1/\omega_i$ term reduces to a 1 for all rows.

$$\begin{bmatrix} E_1(T_5) \\ E_2(T_5) \\ E_3(T_5) \\ E_4(T_5) \\ E_5(T_5) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} \sum_{j \in \mathcal{X}} P_{ij} E_j(T_5) \\ \sum_{j \in \mathcal{X}} P_{ij} E_j(T_5) \\ \sum_{j \in \mathcal{X}} P_{ij} E_j(T_5) \\ \sum_{j \in \mathcal{X}} P_{ij} E_j(T_5) \\ 0 \end{bmatrix} \implies \tag{7}$$

$$E = \begin{bmatrix} E_1(T_5) \\ E_2(T_5) \\ E_3(T_5) \\ E_4(T_5) \end{bmatrix} = \mathbb{1} + P_{(1:4,1:4)} E \implies \tag{8}$$

$$E = (I - P_{(1:4,1:4)})^{-1} \mathbb{1} \tag{9}$$

This reduces to a 4-dimensional (in stead of 5) problem, as $E_5(T_5) = 0$ by definition. Hopefully it is clear that the 5th row and column of P is what becomes redundant in this setup, and of course that $\mathbb{1}$ and I (identity matrix) have the appropriate number of dimensions for this problem. The theoretical hitting time that we were after was $E_1(T_5) = 8.7857$, and we can conclude that the simulated result came close to the theoretical one.

2 - Graph coloring and network games

The simulation was implemented by quite closely following the instructions. The code skeleton for visualization was very helpful. The distributed learning algorithm for potential games was used to solve both problem 2.a) and b). The potential function, cost functions, inverse noise and probability distributions from which the next colors were drawn were all given in the instructions. For the simulations for part two, 10 000 iterations was once again selected as a reasonable ceiling, noting here that in the event of the potential reaching zero before 10 000 iterations are performed, the algorithm should stop.

2.a) Line graph, 2 colors

This task was to color a simple 10-node line graph such that the two colors that were used, red and green, would be evenly distributed on the nodes, i.e. no neighbouring nodes should have the same color. The graph was constructed by

creating the adjacency matrix (simply 10×10 square matrix with zeros except for ones on the off diagonals 1 and -1), and subsequently the normalized weight matrix (as the visualization tool appeared to take that as an input). The result is presented in Figure 1. The potential was minimized down to zero (at which point the colors fulfilled the specifications) already after around 250 iterations. However, what is noteworthy is that the potential function dips down to values 1 or 2 a number of times, only to go up again. The introduction of noise and randomness makes the distributed learning algorithm sometimes move away from "good" solutions, which is in its nature. As $\eta(t) = t/100$ is the encoded noise, I tried making it a bit smaller over time, which resulted in larger noise and that the algorithm took longer to reach optimum, and vice versa when introducing smaller noise.

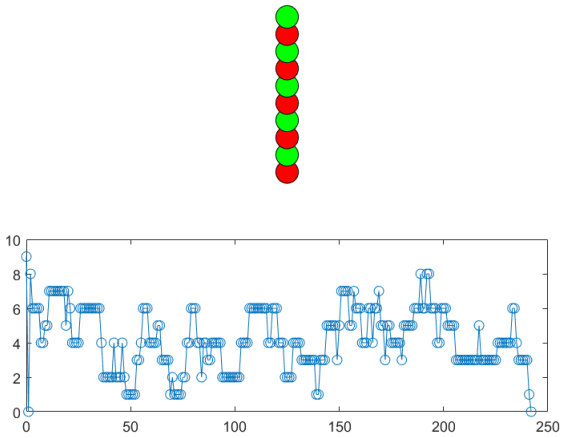

Figure 1: The coloring problem from 2.a)

2.b) Wifi network, 8 colors

This problem was larger, having 100 nodes and 8 colors to take into consideration. The provided data in wifi.mat was the adjacency matrix for the graph, and the normalized weight matrix was computed, adding self-loops for the nodes with no out-degree. The network had some isolated nodes, which made this necessary. Furthermore the cost function was different from 2.a), and was implemented according to the instructions, and the probabilities of the colors were calculated basically identically as in 2.a), differing in that the generation of new color was made using the matlab randsample function. From that, the implementation of the distributed learning algorithm was very much alike the one in 2.a).

The result from the initial simulaton, with the recommended inverse noise is presented in Figure 2, where the final coloring is presented as well as the potential function's evaluation over iterations. The major difference from 2.a) is that even after 10 000 iterations, the potential didn't quite reach zero, but got stuck at between 3 and 5. This is likely to do with the sheer size of the problem, possibly more iterations would remedy this. In Figure 3, I tried with a different inverse noise, namely the constant 0.01, i.e. I didn't allow for the noise to reduce over time. the result is quite apparent, as the algorithm still couldn't find a zero potential with 10 000 iterations, and that the curve for the potential is far less smooth.

The conclusion is that the noise has a big effect on the distributed learning algorithm.

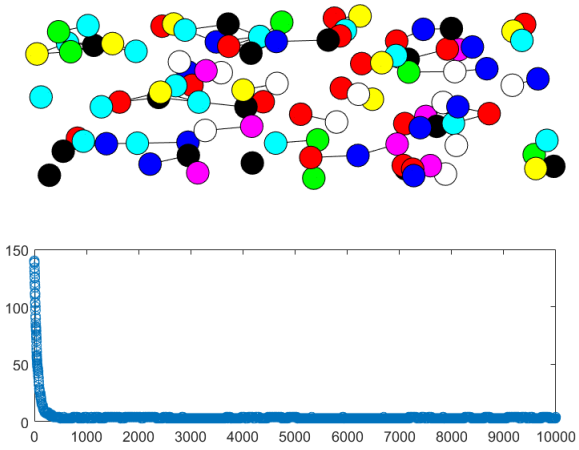

Figure 2: The coloring problem for 2.b) with recommended $\eta(t) = \frac{t}{100}$

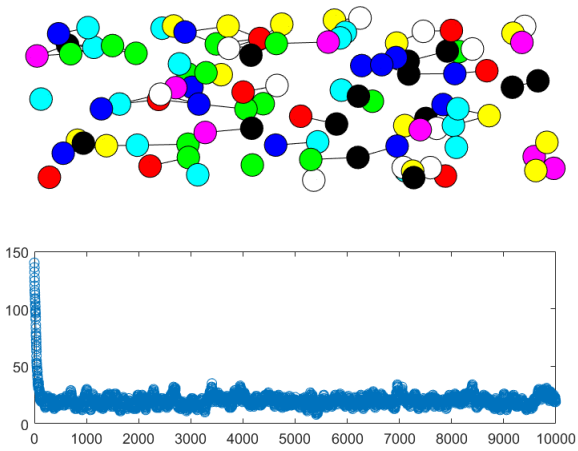

Figure 3: The coloring problem for 2.b) with $\eta(t) = 0.01$, constant