FRTN30 Network Dynamics

Assignment 3

(Van Duy Dang - va7200da-s)

May 16, 2023

1 Single-particle random walk

In this task, we will study a single particle which performs a random walk with the following transition rate matrix.

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

We define $w_i = \sum_f \Lambda_{ij}$. On average, the particle will spend a duration of $1/w_i$ in state i before transitioning to a different state. The normalized weight matrix P is calculated as

$$P = D^{-1}\Lambda \tag{2}$$

where $D = diag(w)$ and $w = \Lambda\mathbb{1}$.

The time S that the particle stays in one state before moving on to the next state is stochastic variable according to

$$\mathbb{P}(S \geq t) = e^{-rt}, t \geq 0, \tag{3}$$

where $r = w_i$ is the rate of the distribution. We find a transition probability matrix $\overline{P}$ by

$$\overline{P}_{ij} = \frac{\Lambda_{ij}}{w_*}, \quad w_* = \max_i w_i \tag{4}$$

$$\overline{P}_{ii} = 1 - \sum_{j \neq i} \overline{P}_{ij} \tag{5}$$

Using the above equations, we calculate

$$\overline{P} = \begin{array}{c} \begin{matrix} \text{o} & \text{a} & \text{b} & \text{c} & \text{d} \end{matrix} \\ \begin{bmatrix} 2/5 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \end{bmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \tag{6}$$

a) Find the average time the particle leaves node 'a' and returns to it

Firstly, we draw a uniform random variable u, $u \in \mathcal{U}(0,1)$. Then we compute t_{next} as

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

We can use *randsample*() from Matlab to randomly select the next position based on the transition probability matrix $\overline{P}$. When the particle returns to node 'a', we save the total

time it takes. After 10^5 iterations, the average time is 6.7335.

b) Compare with the theoretical return-time

Using the chapter 7 from the lecture notes, with $\overline{\pi}$ is an invariant probability vector for the jump chain with transition probability matrix $\overline{P}$, we have

$$\overline{\pi} = \overline{P}'\overline{\pi}$$

$eig()$ from Matlab can be used to find $\overline{\pi}$. Then the expected return times satisfy

$$\mathbb{E}_i[\overline{T}_i^+] = \frac{1}{\omega_i \overline{\pi}_i}, \quad i \in \mathcal{X}$$

where $\overline{T}_i^+ = \inf\{t \geq 0 : X(t) = i \text{ and } X(s) \neq i \text{ for some s} \in (0, t)\}$

After the calculation, we get $\mathbb{E}_a[\overline{T}_a^+] = 6.75$. It seems reasonable because it is close to the result that we computed in the previous task.

c) Find the average time the particle moves from node 'a' to node 'd'

We do the same as task a, but when the next position is node 'd' then we reset the time to 0 and the position to node 'o'. After 10^5 iteration, the average time is 8.7909.

d) Compare with the theoretical hitting time

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

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

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

For the continuous-time case, the expected waiting time for a tick of the Poisson clock is given by $\mathbb{E}(S) = 1/r = 1/\omega_*$. It means that the expectation value before changing to the next state is always the same. In addition, the normalized probability matrix in (8) is $\overline{P}$. We can rewrite the equation (8) to $(I - \overline{P})X = 1$, where X is the expected hitting times matrix. After solving the equation, we can find $\mathbb{E}_o(T_d) = 8.7857$ which is close to the result from task c.

2 Graph coloring and network games

In this task, we will study graph coloring to assign a color to each node in an undirected graph so that no 2 consecutive nodes have the same color.

a) Simulate the learning dynamics

We will study a line graph with 10 nodes. $X_i(t)$ is the i-th node and the set of possible states is $C = \{\text{red, green}\}$. Every node is initialized with red.

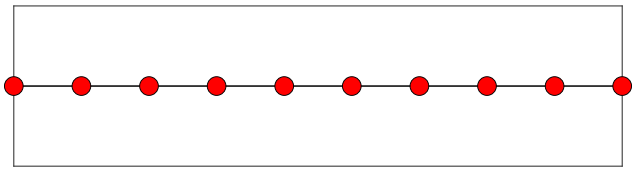

Figure 1: Initialize all nodes to red

One node $I(t)$, chosen uniformly at random, updates its color every time instance t. The new color is chosen from the below probability distribution

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

The cost function is given as

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

We choose $\eta(t) = \frac{t}{100}$. The potential function 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. We loop until the potential is zero. As a consequence, there are no consecutive nodes that have the same color.

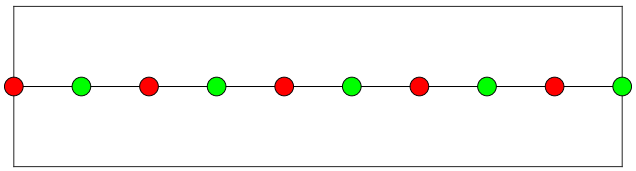

Figure 2: Assign color to nodes

The potential value fluctuates a lot as we choose the node at random. In general, it reaches 0 quite fast. I try multiple times and it takes around 120-300 cycles to find the solution.

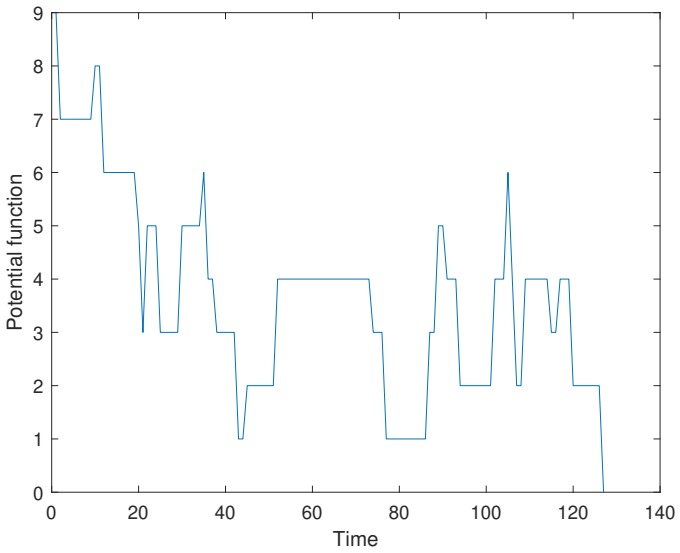

Figure 3: Potential function

b) Assign wifi-channels to routers

In this task, we use the algorithm from the previous task to solve the problem of assigning wifi-channels to routers. The set of possible states, with different colors represent frequency bands, is $C = \{1$: red, 2: green, 3: blue, 4: yellow, 5: magenta, 6: cyan, 7: white, 8: black$\}$. The cost function is given as

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

It means that neighboring routers should avoid using channels with the same frequency band or adjacent frequency bands.

Firstly, I try with $\eta = t/100$.

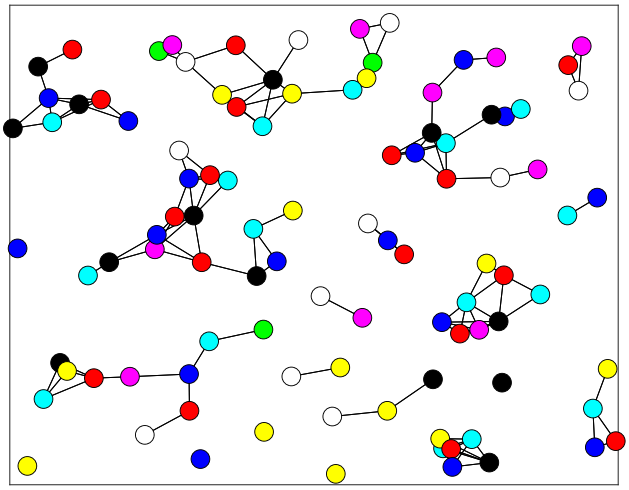

Figure 4: Assign color to nodes with $\eta = t/100$

A near-zero potential solution is found after around 700-800 cycles.

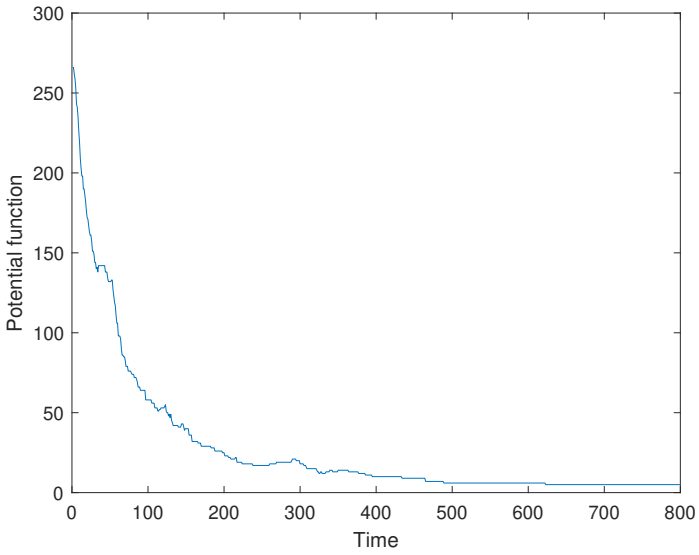

Figure 5: Potential function with $\eta = t/100$

$\eta(t)$ is the inverse of the noise so it may converge faster if we increase $\eta(t)$ to reduce the noise. I try with $\eta = t/10$ and $\eta = t/300$.

$\eta = t/10$ is the best solution that I obtained. It takes around 400-500 cycles to converge.

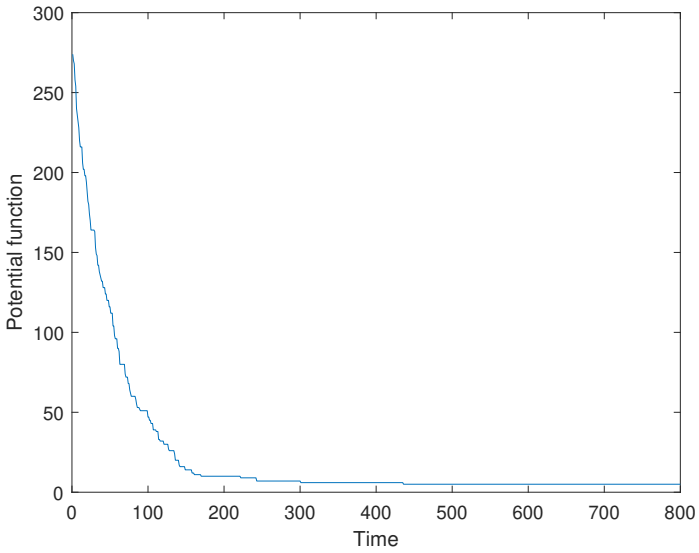

Figure 6: Potential function with $\eta = t/10$

With $\eta = t/300$, the noise is increasing so the potential function fluctuates and the learning algorithm needs more time to converge. In this case, it cannot converge within 800 cycles.

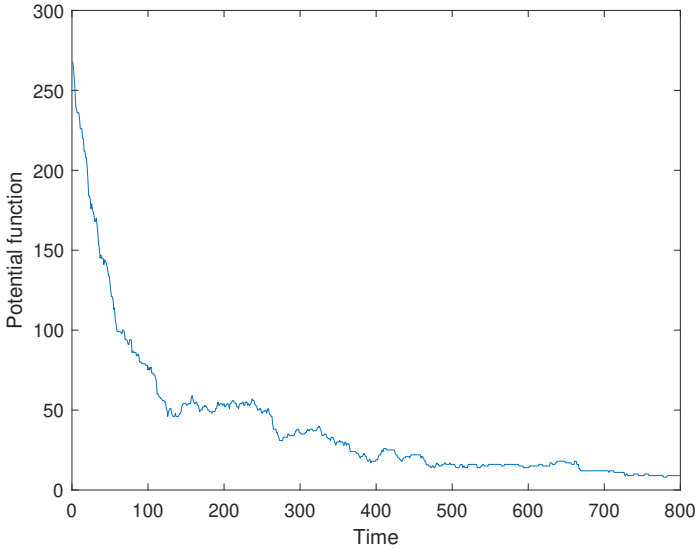

Figure 7: Potential function with $\eta = t/300$

In conclusion, we need to find the appropriate η so that the algorithm converges quickly. If it's too large or too small, the algorithm may take longer or never converge.