FRTN30 Network Dynamics: Hand-In 3

Alfred Bornefalk, al1718bo-s@student.lu.se, 2022-06-03

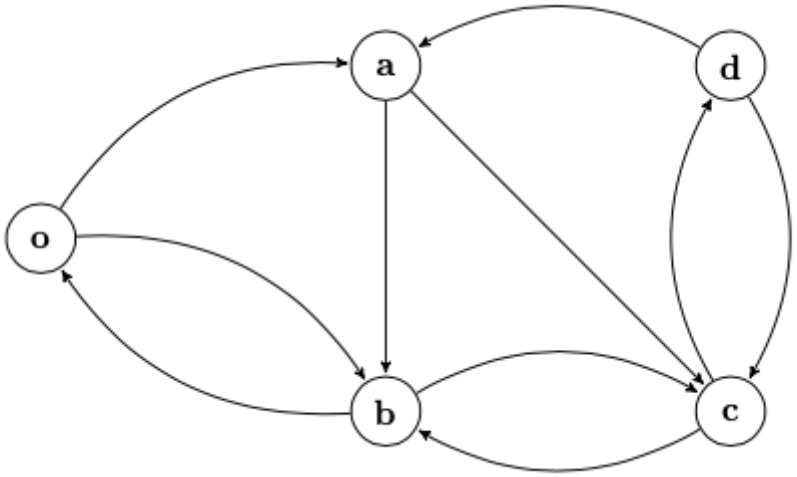

Table of Contents

1. Single-Particle Random Walk **3**
1.1. The Return Time 3
1.1.1. The Simulated Average Return Time 3
1.1.2. The Theoretical Average Return Time 4
1.2. The Hitting Time 5
1.2.1. The Simulated Average Hitting Time 5
1.2.2. The Theoretical Average Hitting Time 5

2. Graph Coloring and Network Games **6**
2.1. Line Graph With Ten Nodes 6
2.2. Assigning WiFi-Channels to Routers 10

1. Single-Particle Random Walk

In the first part of this assignment, we are going to investigate the random walk of a single particle. The particle in question moves around in a closed network consisting of the nodes *o*, *a*, *b*, *c*, and *d*, and its transition matrix Λ is given below.

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

When reading table 1, 2/5 on row 1 and column 2 means that the probability for the particle to move from *o* to *a* in the next step is, indeed, .40. We will simulate the particle moving around in the network in continuous time according to the matrix properties in table 1.

1.1. The Return Time

In this subsection, we are asked to compute the average time it takes a particle that starts in node *a* to leave that node and then return to it. This is equivalent to calculating the average return time for node *a* in our given network. In plain words, the return time T_j^+ is defined as the first time $t \geq 1$ that the Markov chain $X(t)$ hits j. More generally, with the transition matrix Λ on a finite state space χ, for a subset of states $S \subseteq \chi$, the return time is defined as

$$T_S^+ := inf\{t \geq 1 : X(t) \in S\} = min_{s \in S}\{T_S^+\}$$

1.1.1. The Simulated Average Return Time

First, we will simulate the average return time for the particle starting in node *a*. We begin with calculating the out-degree w on each node. Thereafter, we create the diagonal matrix D, whose diagonal entries are equal to 1 subtracted by the out-degree of the corresponding node. Finally, we compute the normalized transition matrix, i.e.

$$Q = D + \Lambda$$

For illustration purposes, the obtained matrix Q is presented below.

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

As can be seen in Q, the out-degree is now 1 for every node. Moving on with the fact that 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

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

where $r = \omega_i = \sum_j \Lambda_{ij}$ is the rate of the distribution, we can simulate the random walk for the single particle.

For cost efficiency purposes, whilst still having a large enough sample, we let the single particle take 1,000,000 steps, and store all of these steps in an array. From this, we create a subarray containing all the time values regarding how long it has taken for the single particle to start in, and return to, a. Taking the average of the subarray in question, we finally get that the simulated average return time is equal to

$$T_a^+(simulated) = 6.7633$$

1.1.2. The Theoretical Average Return Time

In order to investigate the validity of the simulated average return time, we are now going to calculate the theoretical average return time. To begin, we compute the eigenvector of Q. After controlling that the sum of the entries does not equal 0, we normalize said eigenvector, thus yielding the invariant distribution π for our network. To obtain the theoretical average return time, we simply calculate

$$T_a^+(theoretical) = \frac{1}{\pi_a}$$

This results in a value of 6.7500; the simulated value is a good approximation of the theoretical one, and the difference between the two would (expectedly) be even smaller if we were to let the particle take more than 1,000,000 steps.

1.2. The Hitting Time

In this subsection, we are asked to compute the average time it takes a particle that starts in node *o* to end up in node *d*. This is equivalent to calculating the average hitting time for node *d* when starting in node *o*. In plain words, the hit time T_j is defined as the first time $t \geq 0$ that $X(t)$ hits j. More generally, like previously, with Λ on a finite state space χ, for a subset of states $S \subseteq \chi$, the hitting time is defined as

$$T_S := inf\{t \geq 0 : X(t) \in S\} = min_{s \in S}\{T_S\}$$

As can be seen, this is very similar to the case of return time, with the major difference being that $t_{hitting} \geq 0$, whilst $t_{return} \geq 1$. This comes from the fact that the former case measures how long it takes for $X(t)$ to move from a state i to another state j, whereas the latter instead measures how long it takes to get back to state i. Consequently, both are examples of stopping times, and the mechanics of deriving them are similar, which we will see below.

1.2.1. The Simulated Average Hitting Time

The only difference in this case compared to the simulation of the return time, is that we instead want to consider the average time it takes to get from *o* to *d*. Again, we perform a walk consisting of 1,000,000 steps, and store these in an array. Then, we create a subarray by iterating through the randomized walk that has taken place, storing all the time values corresponding to starting in *o* and ending up in *d*. Once again, we take the average of the elements in the subarray, and get that the simulated average hitting time is equal to

$$T_{od}(simulated) = 8.8095$$

1.2.2. The Theoretical Average Hitting Time

Again, we wish to compare the simulated derived time above with the theoretical one. We do so by first manipulating Q such that the node *d* is excluded, yielding the matrix shown below.

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

Denoting the matrix above with Q^{MOD}, we know from proposition 7.2 in the lecture notes of the course that we can proceed by the taking the inverse of the difference between the unit matrix and Q^{MOD}, i.e.

$$(I - Q^{MOD})^{-1}$$

Thereafter, we multiply this result with an array containing only ones in order to get the probability distribution π^{MOD} of going from a node *o*, *a*, *b*, or *c*, and ending up in node *d*. Since we are only interested in the hitting time from *o* to *d*, the final calculation required is simply

$$T_{od}(theoretical) = \frac{1}{\pi_o^{MOD}}$$

This results in a value of 8.7857, meaning that the simulated value is close to the theoretical one. Once more, the difference between the two would (expectedly) be even smaller if we were to let the random walk proceed even longer.

2. Graph Coloring and Network Games

In the second, and final, part of this assignment we will study graph coloring as an application of distributed learning in potential games. The very aim of graph coloring is to assign a color to each node in a given undirected graph, such that none of its neighbors have the same color. First, we will examine a simple line graph in order to illustrate the distributed learning algorithm. After that, we are going to look at a more general example, in which non-interfering channels to WiFi access points will be assigned.

2.1. Line Graph With Ten Nodes

To get acquainted with the learning dynamics, we begin with examining a line graph with ten nodes, where the i^{th} nodes' state is denoted by $X_i(t)$; the set of possible colors consists of $C = \{\text{green, red}\}$. At initialization, every node is red, and at every time instance t a uniformly random chosen node wakes up and updates its color according to the probability distribution

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

where the cost function is 1 if $X_j(t) = s$, and 0 otherwise. The inverse of the noise is set to

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

To assess how close to a solution the learning algorithm is (i.e., how close it is to assign colors in such a way that none of the nodes have any neighbors with the same color), the potential function U is evaluated according to

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

By construction, $U = 0$ means that there are no conflicting nodes; a solution is found.

To simulate the learning dynamics above, we choose to compute 100 iterations (updates). Before the first update, all of the nodes are set to red, as can be seen in figure 1 below.

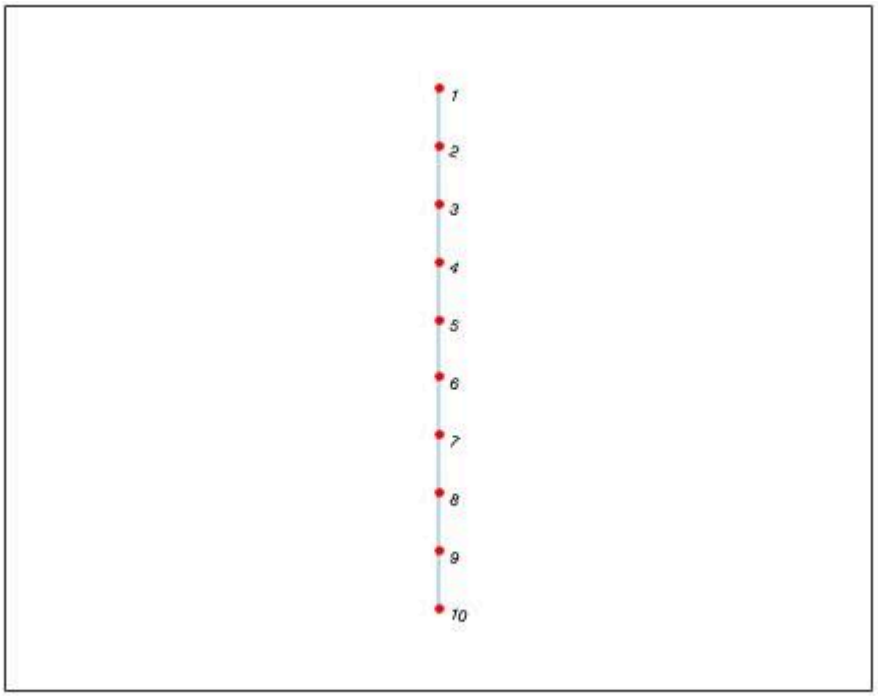

Figure 1: Initial state of the line graph.

Then, at $t = 1$, a node is drawn uniformly and updated according to P, which, aside from $\eta(t)$, takes into consideration the amount of red and green neighbors the node has. Letting $P = 1$ mean that the probability of updating to a green node is equal to 100% (and $P = 0$ conveying a 0% probability of updating to a green node), we then draw a random number uniformly between 0 and 1. If it is less than P, the node in question updates to green, and vice versa. We do this for $t = 2, 3, \ldots, 100$, and then calculate the potential function. Since the graph coloring has a degree of randomness to it, the value of U differs depending on the simulation performed. Figure 2 displays a simulation where $U = 2$, whereas figure 3 showcases a scenario in which $U = 4$ after 100 updates.

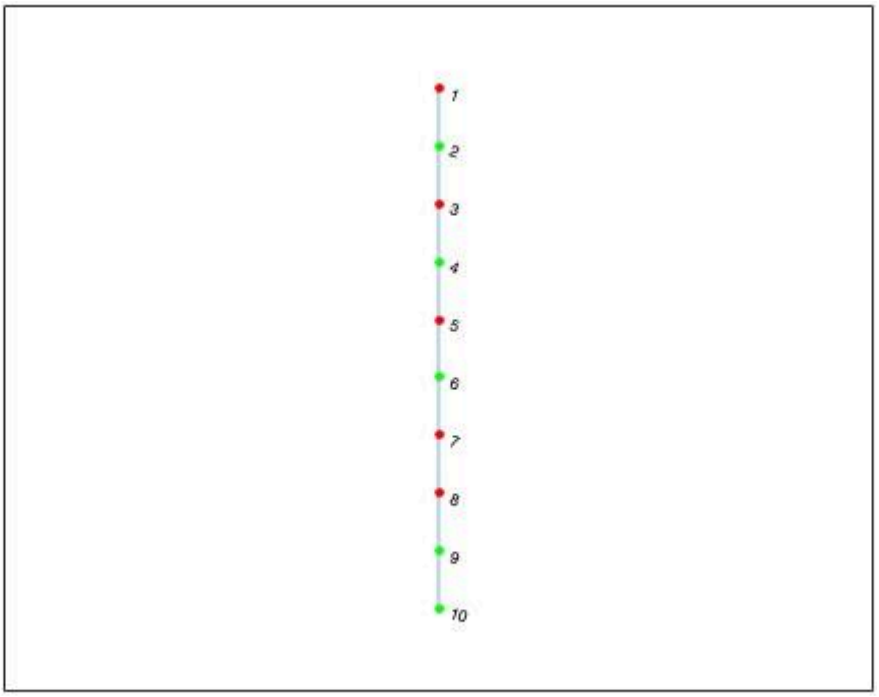

Figure 2: Simulation with 100 iterations resulting in $U = 2$ *for the line graph.*

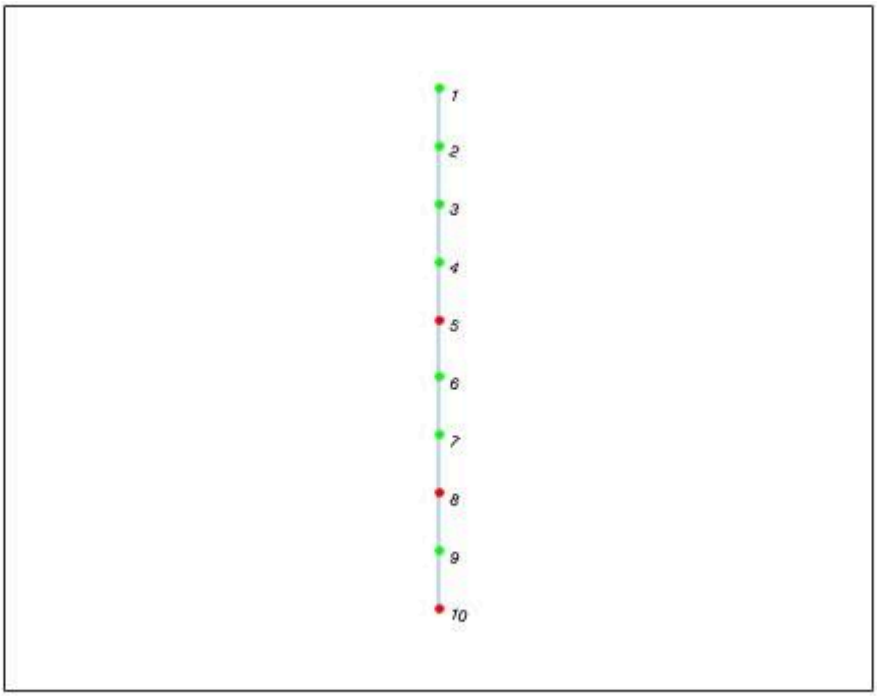

Figure 3: Simulation with 100 iterations resulting in $U = 4$ *for the line graph.*

Examining figure 2 and figure 3, it is evident that the former is closer to the defined solution of the graph coloring problem. The total cost in figure 2 is 4, whilst the total cost in figure 3 is 8; the simulated learning dynamics worked better in the first case. Still, none of the total costs are equal to 0. In order to examine if the line graph ever converges to this state, we increase the number of updates to 500. The result from this is displayed below in figure 4.

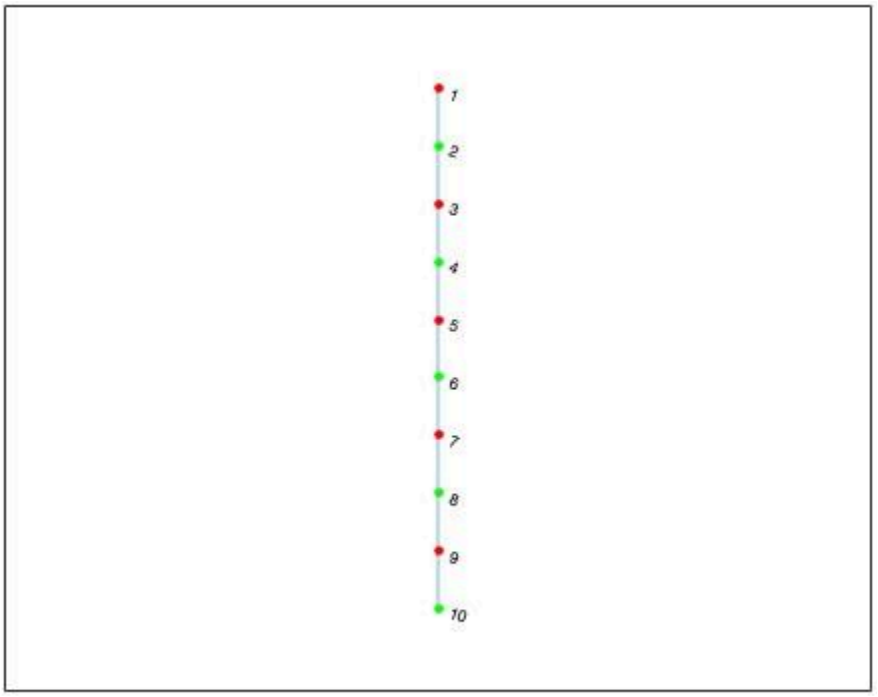

Figure 4: Simulation with 500 iterations resulting in $U = 0$ for the line graph.

As can be seen in figure 4, we have now obtained an optimal solution, in which the total cost of the line graph is 0. To understand the behavior of the learning dynamics, we plot the potential function in figure 5.

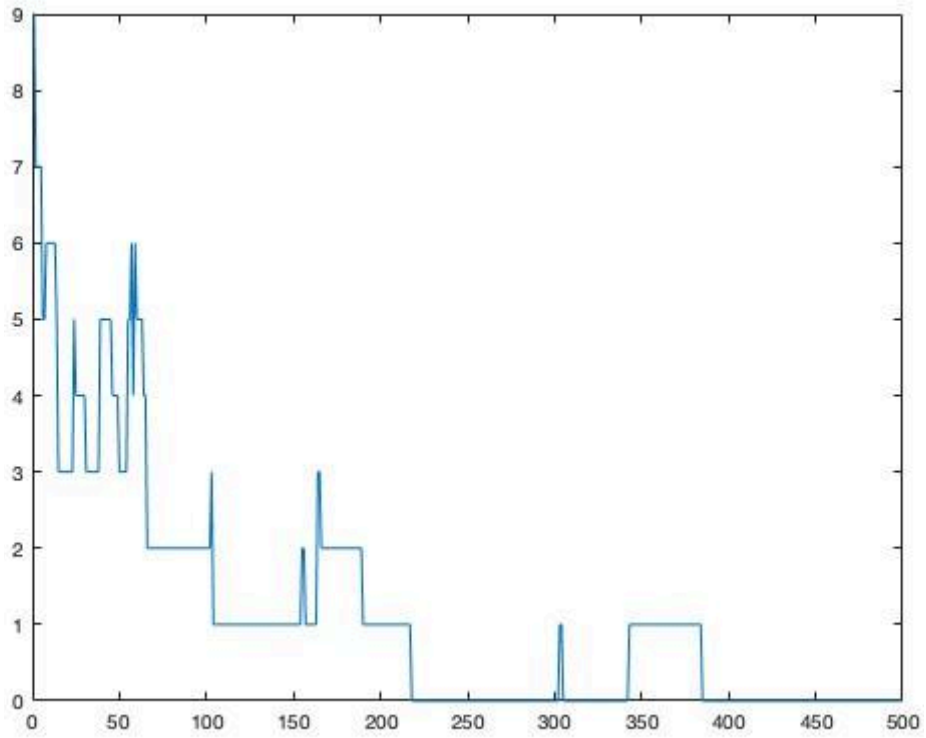

Figure 5: Plot of the potential function for the line graph. The x-axis represents the total number of updates, and the y-axis represents the value of the potential function.

In figure 5, we can observe that the potential of the line graph starts at 9. After only a few updates, it reaches a value of 6, which it never exceeds from here. After circa 220 updates, it hits the value 0 for the first time. In two periods, the first one shorter and the second one longer, it reverts back to a potential of 1. After approximately 380 updates, it reaches 0 once again, and stays at this solution until the very end of the simulation. With the acquired intuition on learning dynamics and the potential function, we can now proceed to the last subsection of this assignment.

2.2. Assigning WiFi-Channels to Routers

At last, we will use the coloring algorithm to assign WiFi-channels to routers, where the adjacency matrix W of a network of 100 routers is given in a separate file. In this case, a link between two nodes means that the two corresponding routers are able to interfere with each other. This time around, the set of possible colors is given by $C = \{$red, green, blue, yellow, magenta, cyan, white, black$\}$, with the colors representing frequency bands. The cost function is 2 if $X_j(t) = s$, 1 if $|X_j(t) - s| = 1$, and 0 otherwise. The interpretation of this cost function is that a router is punished (associated with higher cost) if it uses channels with the same frequency band, or a frequency band right next to it.

In our simulation, we want to verify that a near-zero potential solution is found for the router network after a sufficient number of iterations (or updates) when using $\eta(t) = \frac{t}{100}$. In addition, we aim to observe what happens when trying some other functions of $\eta(t)$. To examine the impact of the choice of the inverse of the noise function, we will simulate four different cases: $\eta_1(t) = \frac{t}{10{,}000}$, $\eta_2(t) = \frac{t}{1{,}000}$, $\eta_3(t) = \frac{t}{100}$, and $\eta_4(t) = \frac{t}{10}$. Regardless of the choice of $\eta(t)$, the router network has the initial state of every router being red (using the same frequency band); this is visualized in figure 6.

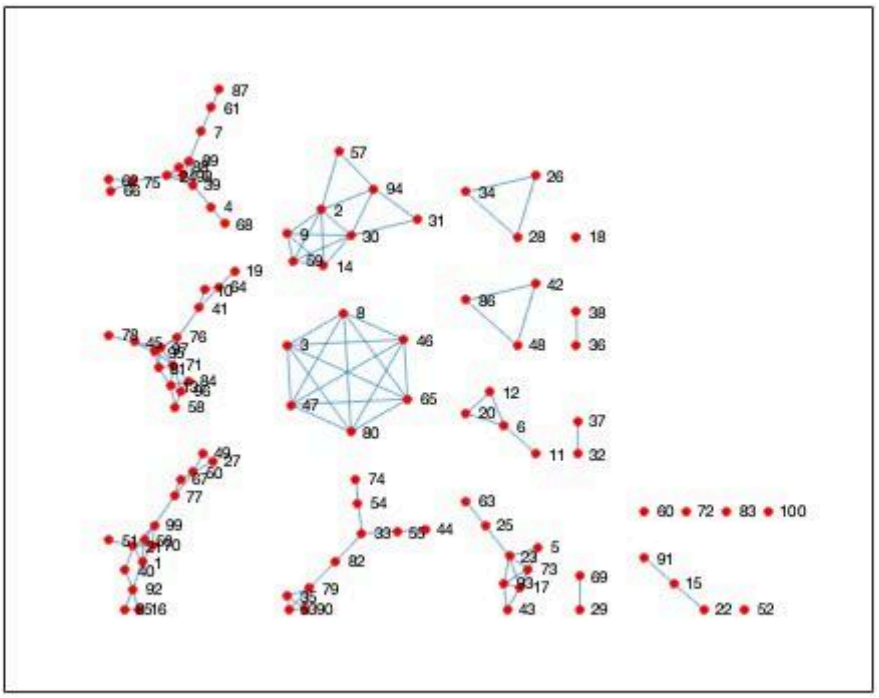

Figure 6: Initial state of the network of routers.

To simulate the learning dynamics for each choice of $\eta(t)$, we choose to perform 1,000 iterations. Then, at $t = 1$, a node is drawn uniformly and updated according to P, just like in the previous case. (The difference being the current choice of the inverse of the noise, and how the cost function is defined.) The probabilistic nature of the problem means that a higher (lower) P is associated with a higher (lower) probability of updating to black (represented by 8 in MATLAB), and so on. Continuing to do this for $t = 2, 3, \ldots, 1,000$, we then perform a final calculation of the total cost of the router network. Multiplying this value with .50 (or, equivalently, dividing it by 2), yields U for the simulation at hand. Below, figure 7, figure 8, figure 9, and figure 10 represents simulations for $\eta_1(t)$, $\eta_2(t)$, $\eta_3(t)$, and $\eta_4(t)$, respectively.

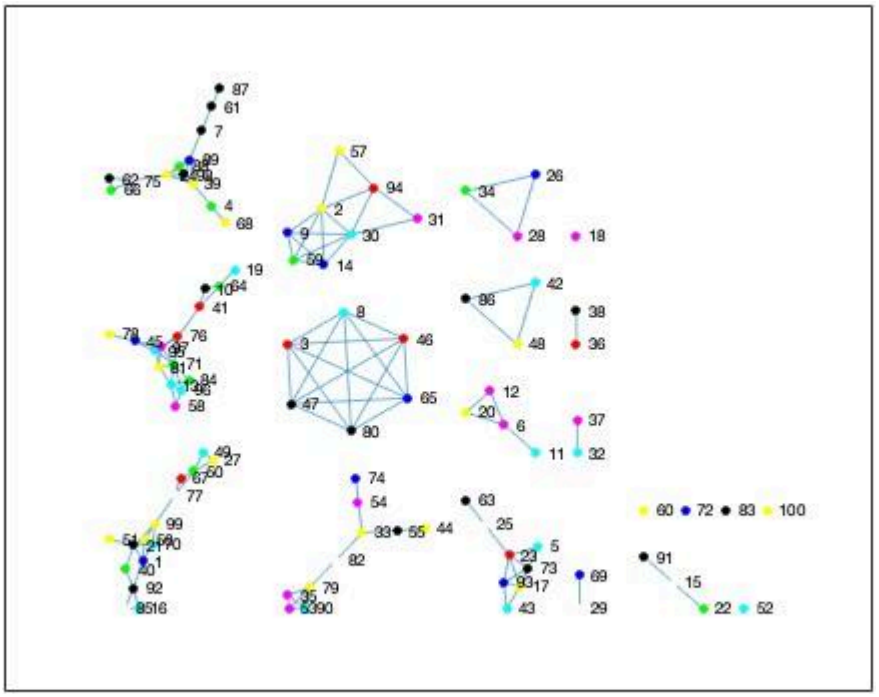

Figure 7: Simulation with $\eta_1(t)$ *resulting in* $U = 59$ *for the network of routers.*

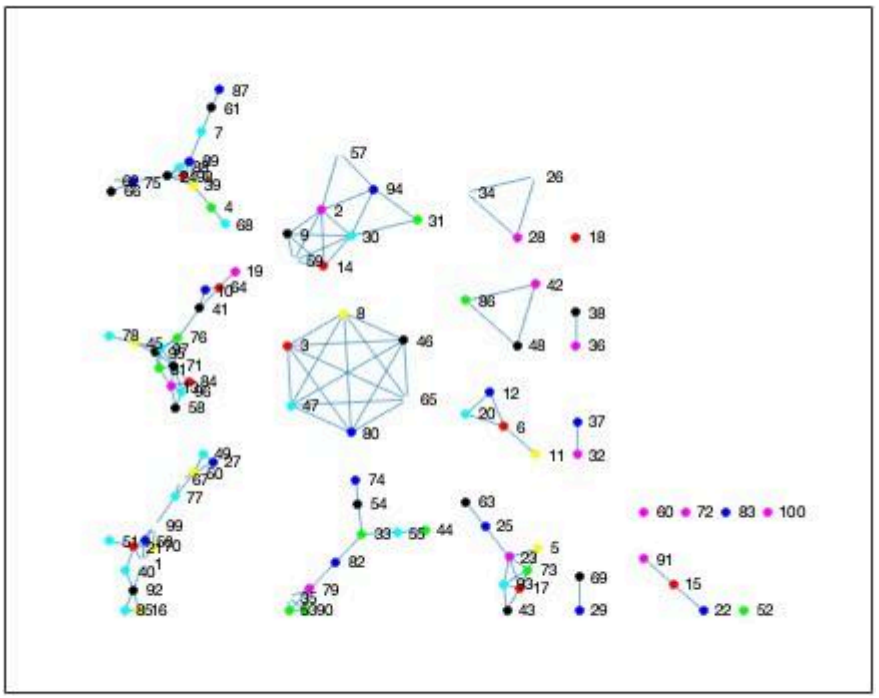

Figure 8: Simulation with $\eta_2(t)$ *resulting in* $U = 25$ *for the network of routers.*

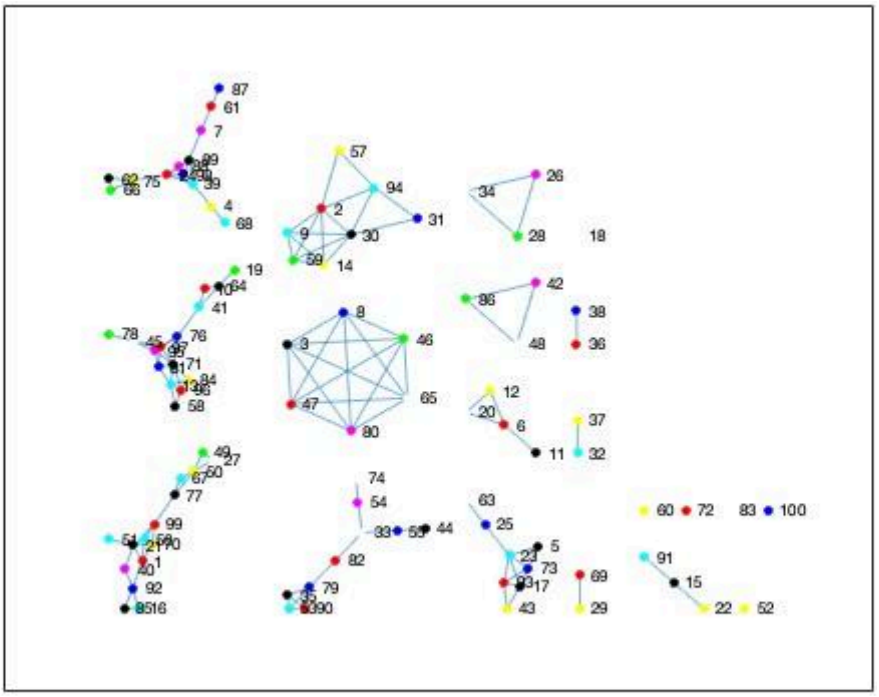

Figure 9: Simulation with $\eta_3(t)$ resulting in $U = 4$ for the network of routers.

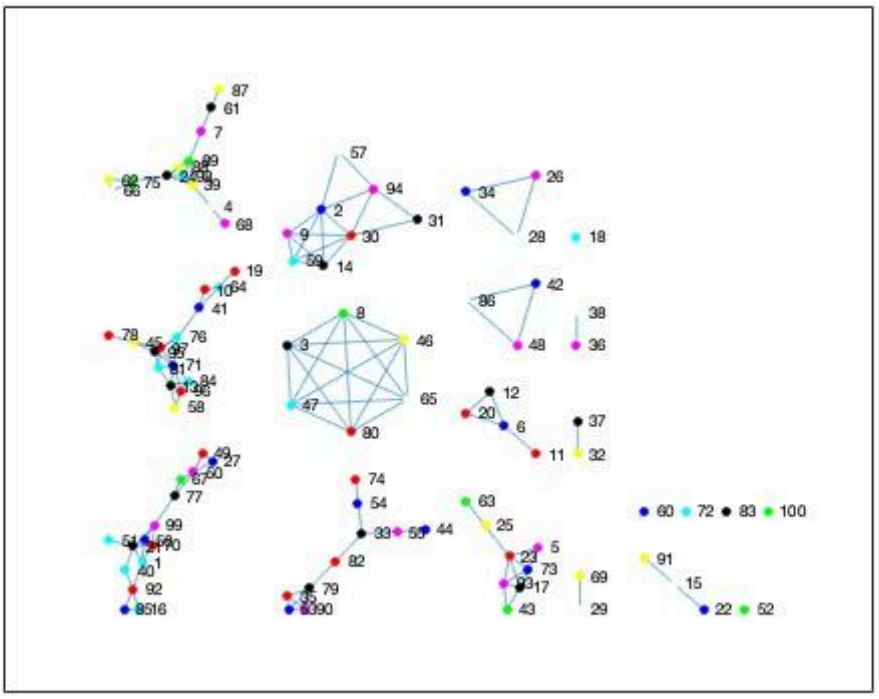

Figure 10: Simulation with $\eta_4(t)$ resulting in $U = 4$ for the network of routers.

Examining figures 7 through 10, we can draw several conclusions. Firstly, it is now verified that η_3, with an associated potential of 4 in the simulation, yields a near-zero potential solution for our network. Secondly, the plots strongly suggest that increasing $\eta(t)$ has an impact on the potential, but only up to a certain point. Figure 7 shows that $\eta_1(t)$ resulted in

$U = 59$; figure 8 shows that this potential was more than cut in half with $\eta_2(t)$. Moving on to $\eta_3(t)$, figure 9 showcases that U decreases all the way from 27 to the near-zero value of 4. However, as can be seen in figure 10, going from $\eta_3(t)$ to $\eta_4(t)$ did not bear any effect on the potential in the simulation. Perhaps not too surprisingly, the function provided by the people responsible for this class works well in this section. However, it is possible, and perhaps even plausible, that the potential of the network of routers decreases faster with $\eta_4(t)$ than $\eta_3(t)$. To examine the effect over time of the choice of $\eta(t)$ on the learning dynamics, we perform a new simulation for each inverse of the noise, and plot the potential functions for the four different cases; this is presented in figure 11 below.

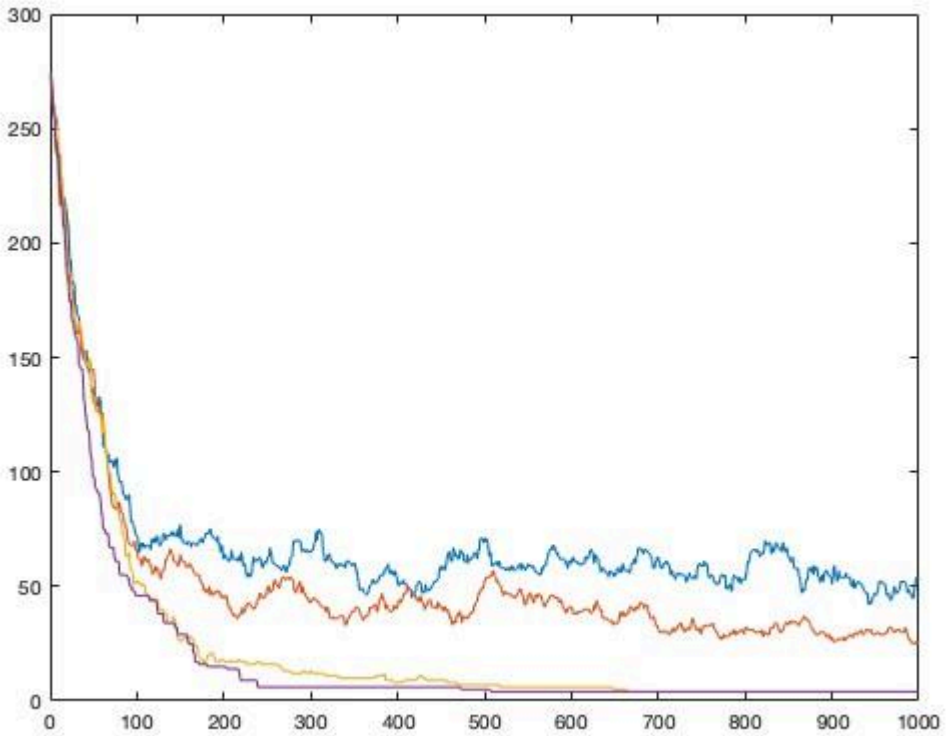

Figure 11: Plot of the potential function for the network of routers. The x-axis represents the total number of updates, and the y-axis represents the value of the potential function. The blue, red, yellow, and purple plot represent $\eta_1(t)$, $\eta_2(t)$, $\eta_3(t)$, *and* $\eta_4(t)$, *respectively.*

As can be seen in figure 11, both $\eta_3(t)$ than $\eta_4(t)$ converge to a potential of 4. However, the rate of convergence for the latter is faster, reaching (and staying in) a configuration with the potential 4 before 500 updates, whereas it takes the former around 650 updates to obtain the best solution. Examining $\eta_1(t)$, we can see that it has a weak convergence rate, with close to no improvements of the potential function after the last update compared to after the 200th. $\eta_2(t)$ displays a better learning dynamic, but it is still likely that it will have to keep updating for a considerable amount of time before reaching the value 4. Overall, $\eta_4(t)$ seems to be the best choice; fast convergence is essential in problems requiring large computational power.

To conclude, we will take notice of the relevance of subject matters like these for people's everyday lives. Whether it be doing research on the Internet at school, or joining a meeting at work, we expect our connection to be stable. From time to time though, one might attend a large gathering such as a game at The Big House in Michigan, and then get equally surprised and irritated that the network is seemingly down. The graph coloring simulations we have completed in this assignment illustrates the complexity of assigning traffic to unoccupied frequency bands; along with other factors, one can reminisce that a possible reason behind a lagging connection might be the choice of the inverse of the noise.