Single-Particle Random Walks and Network Games

Nils Hallerfelt

Maj 2023

1 Single-Particle Random Walk

We will study single-particle dynamics in a graph, where the dynamics are given by a transition rate matrix Λ:

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

From the given transition rates a single particles movements will be simulated in continuous time. It is assumed the particle dynamics has the Markov property, and to define the continuous-time Markov chain a Poisson process is used as follows. For a given r (rate), let $S_1, S_2, ...$ be a sequence of independent random variables with r-exponential distribution:

$$\mathbb{P}(S_i \geq t) = \exp(-rt), \quad t \geq 0$$

The exponential distribution is memoryless, so for any event S_i that occurs, the time for the next event to occur is an independent exponential random variable (independent from previous events that has occurred):

$$T_0 = 0, \quad T_k = \sum_{1 \leq j \leq k} S_j, \quad k = 1, 2, ...$$

Then the Poisson process can be defined as

$$N_t = \sup\{k \geq 0 : T_k \geq t\}$$

where N_t then stands for the number of events occurred by time t.

Here a $\omega*$-Poisson clock will be used, where $\omega*$ is the maximum degree of the degrees in Λ, which corresponds to the shortest waiting time in the nodes given the Λ-transition rates. Then $\bar{P}$ transition probability matrix is defined as:

$$\bar{P}_{ij} = \frac{\Lambda}{\omega*}, \quad \bar{P}_{ii} = 1 - \sum_{j \neq i} \bar{P}_{ij}$$

With these transition probabilities and the Poisson clock we can define the jump chain $X(t) = U(N_t)$, a discrete Markov chain of the continuous-time Poisson process that, in our case, is the chain of states states in continuous time that the particle visits.

1.1 Expected Return Time

To find the average (expected) simulated return time, we begin by defining the analytical (expected) return times as (Lecture notes on Network Dynamics, Giacomo Como and Fabio Fagnani):

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

where $\bar{\pi}$ is the stationary distribution of the continuous time Markov chain. To find the the stationary distribution, one can use the following (Lecture notes on Network Dynamics, Giacomo Como and Fabio Fagnani):

$$\bar{P}'\bar{\pi} = \bar{\pi} \iff (\bar{P}' - I)\bar{\pi} = 0$$

Thus we are interested in the null-space of the matrix $\bar{P}' - I$, which easily can be found. If the $\bar{P} - I$ is off full rank, the null-space is empty and a stationary distribution does not exist. However this did not pose a problem, as $\bar{P}' - I$ is non-invertible. As the null-space is a space spanned by a number of basis vectors, any linear combination of these will give a solution. However, as the Λ induces a $\bar{P}' - I$ null-space consisting of only one basis vector, the following constraint is sufficient to find a unique solution:

$$\bar{\pi}\mathbb{1} = \mathbb{1} \implies \bar{\pi} = [0.1852, 0.1481, 0.2222, 0.2222, 0.2222]'$$

Then the expected return times can easily be determined using eq 1:

$$\mathbb{E}[\bar{T}_a^+] = \frac{1}{\bar{\pi}_a \omega_a} = \frac{1}{0.1481 \cdot 1} = 6.7522$$

To determine the expected return time from simulation, eq 1 is used once again, with the difference that $\bar{\pi}$ is estimated by simulation. This is done by simulating a continuous ω^* Poisson process for the time, and the connected discrete jump-Markov chain $U(N_t)$. From this, $\bar{\pi}^{est}$ can be computed as the relative time spent in each state of the chain $U(N_t)$. This resulted in the following, using a chain of 1000000 steps:

$$\bar{\pi}^{est} = [0.1848, 0.1484, 0.2222, 0.2222, 0.2222]' \tag{2}$$

Then the simulated expected return time is computed:

$$\mathbb{E}[\bar{T}_a^+] \approx \frac{1}{\bar{\pi}_a^{est} \omega_a} = \frac{1}{0.1484 \cdot 1} = 6.7385 \tag{3}$$

As we can see the analytical and the estimated return times are similar, as a consequence of the fact that the estimated and analytical stationary distributions are similar. This is a consequence of the following (Lecture notes on Network Dynamics, Giacomo Como and Fabio Fagnani):

$$\lim_{t \to \infty} \bar{\pi}(t) = \bar{\pi}$$

1.2 Expected Hitting Times

Now we are to determine the expected hitting times, i.e. the expected time it takes for the random process to travel from an origin node to o a destination node d or more in general a set S of destination nodes d_i. To calculate the analytical hitting times we define the matrix P as: $P = diag(\omega)^{-1}\Lambda$. Then the expected hitting times can determined as follows (assuming that the graph in question is connected) (Lecture notes on Network

Dynamics, Giacomo Como and Fabio Fagnani):

$$\bar{\tau}_i^S = \mathbb{E}[T_S], \quad i \in \mathcal{X}$$

are the unique solution of

$$\bar{\tau}_s^S = 0, \quad s \in S, \quad \bar{\tau}_i^S = \frac{1}{\omega_i} + \sum_{j \in \mathcal{X}} P_{i,j} \bar{\tau}_j^S, \quad i \in \mathcal{X} \backslash S$$

For our purposes we will use that $S = \{d\}$. From the above equation we can derive t he following way to compute the hitting times of all nodes not in S to S, where the notation $P_{\backslash S}$ is the matrix P where each row and columns corresponding to S have been removed:

$$\bar{\tau}^S = (I - P_{\backslash S}) \frac{1}{\omega} \mathbb{1}$$

By taking $S = \{d\}$, $\bar{\tau}^S$ holds the hitting times for all nodes except d, to d. This gives the result:

$$\bar{\tau}_o^d = 8.7857$$

To approximate the hitting times, a simulation can be run multiple times, starting in o and stopping when reaching the destination d. Then the average time to reach d from o among these simulation will be a simulated hitting time. With 100000 simulations, the simulated hitting time became 8.7867.

2 Graph Colouring and Network Games

Graph coloring is a method of assignment of labels (called colors) to the vertices of a graph G such that no two adjacent vertices have the same color. It is a special case of graph labeling, which is a way to assign labels to the edges or vertices, or both, of a graph. Here we will study vertex colouring in an undirected graph such that none or as few as possible of each vertices neighbors have the same color as that vertex.

A distributed algorithm will be deployed, which works the following way:

- Each node in the graph is assigned a color randomly from a given color set.
- On each step of the algorithm, a node is selected randomly. It calculates the cost of changing its color to each available color, given a defined cost function.
- The node chooses a new color based on the calculated costs and a probability distribution. For example a probability distribution where the probability of choosing a color is inversely proportional to the cost could be used. This introduces additional randomness into the algorithm, allowing it to potentially escape local minima of the potential function.
- After the node has potentially changed its color, the potential function is updated. If the node did change its color, this will affect the potential function's value. The potential function should be defined such that something that is closer to an optimal solution gives lower potential, and vice versa.
- The algorithm checks if the potential function's value is below some threshold. If it is, that means the graph is properly (optimal or otherwise good enough according to the threshold) colored, and the algorithm terminates. If not, it goes back to the iterative update step.

3 A Line Graph with Binary states

First we will evaluate the algorithm on a simple line graph with 10 vertices, and only two colours (1 and 0). Let the set of states be $S = \{0, 1\}$ and the state of graph at time-step t be represented by a vector $X(t)$, where $X_i(t)$ is the state of node i at time-step t. The cost function is defined as follows:

$$c(s, X_j(t)) = \begin{cases} 1 & \text{if} X_j(t) = s \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The change point probability distribution is a Boltzman-ditribution defined as:

$$P(X_i(t) = a | X(t), i) = \frac{\exp\{-\eta(t) \sum_j A_{i,j} c(a, X_j(t))\}}{\sum_{s \in S} \exp\{-\eta(t) \sum_j A_{i,j} c(s, X_j(t))\}} \tag{5}$$

Here η appears. This function will affect the change colour detection so that it's behaviour varies in time, allowing us to decide how the distribution evolves. The $\eta(t)$ term is a parameter that can be thought of as the inverse of a temperature in a physical system. When $\eta(t)$ is high (i.e., the "temperature" is low), the system is more likely to be in a state of lower energy (cost). When $\eta(t)$ is low (i.e., the "temperature" is high), the system is more likely to explore states of higher energy (cost). This allows the system to potentially escape local minima of the cost function. For this problem η is defined as:

$$\eta(t) = \frac{t}{100} \tag{6}$$

In general, the value of $\eta(t)$ should be high (equivalent to a low temperature) at the end of the algorithm to help it converge to a minimum, and low (high temperature) at the start to allow exploration of the solution space. This mirrors the concept from the field of optimization under physical entropy-like conditions, where you start 'hot' and 'cool' down the system over time to settle in a state of minimal energy. This is much like how a Boltzmann machine works in the context of neural networks and probabilistic unsupervised (generative) machine learning.

The potential function is defined in a natural way:

$$U(t) = \frac{1}{2} \sum_{i,j \in \mathcal{V}} A_{i,j} c(X_i(t), X_j(t)) \tag{7}$$

i.e. in terms of the sum of the cost between all nodes in the graph. Thus, if the potential function is equal to zero, a perfect colouring has been reached.

By running the algorithm with the choice of cost function from 4, inverse noise function from 6, change distribution from 5, and potential from 7 a perfect colouring was found in every run. To summarise the result, the algorithm was run to completion 10000 times, and for each run the evolution of the potential function was recorded. An expected value was then calculated (as the mean from each time-step in the algorithm). This is visualized in figure 1, which also reveals a logarithmic relationship between the potential and the number of time-steps.

3.1 Assigning wifi-channels to Routers

Here, a link between two nodes means that the two routers are able to interfere with each other. The set of possible states now is $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$. This issue is more complex as there are multiple states and a

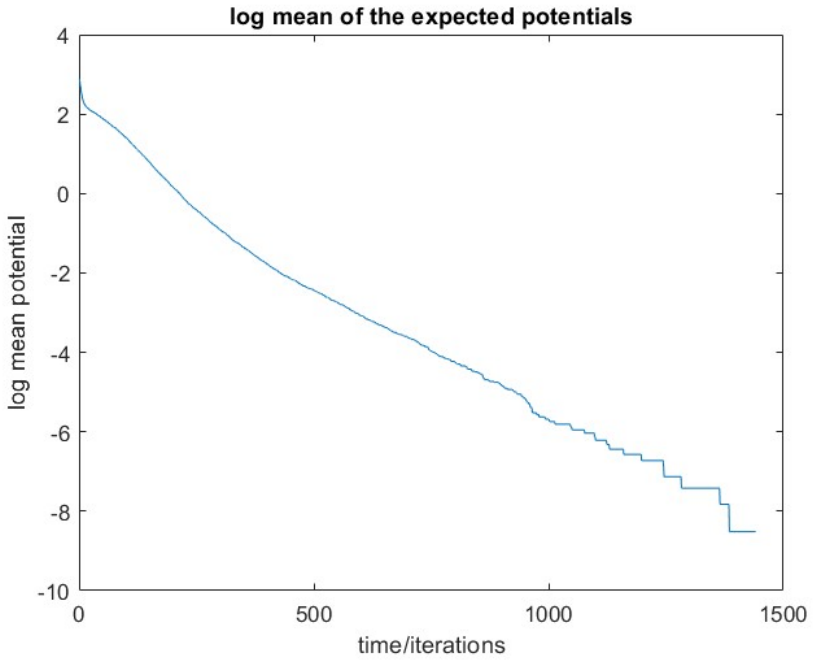

Figure 1: Averaging of the potential function $U(t)$ as a function of the time-steps over 10000 runs. The plot is in logarithmic scale for illustration purposes, but also reveals a decreasing logarithmic relationship between the number of iterations and the potential.

larger network (100 vertices). The graph in question can be seen in figure 2.

The cost function is also altered:

$$c(s, X_j(t)) = \begin{cases} 2 & \text{if } X_j(t) = s \\ 1 & \text{if } |X_j(t) - s| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

to 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.

A number of different functions η were evaluated. First, $\eta = t/100$ was used. When running the algorithm, it was found that a solution with lower potential than 4 could not be found.

It is impossible to get a colouring such that the potential is zero on the subgraph in 3, with this cost function. One can easily see that any optimal colouring of this subgraph gives a potential of 3. Two possible optimal solutions are for example $[1, 1, 3, 5, 7, 8]$ and $[1, 3, 4, 5, 6, 8]$. In addition to this, there exist another subgraph that does not seem to have an optimal colouring with potential lower than 1, however this subgraph is larger and not complete, making it harder to prove the optimality of this potential. However multiple trials on this subgraph could not find any solution with potential lower than 1.

Running the algorithm 100 times, and taking the average potential in each time-step of each run, resulted in the following plot (figure 4):

As we can see from the plot the algorithm finds a close-to-optimal solution relatively quickly, but in some runs it get stuck for a long while in the later stages of the optimization. Therefore it was tried to use a slower

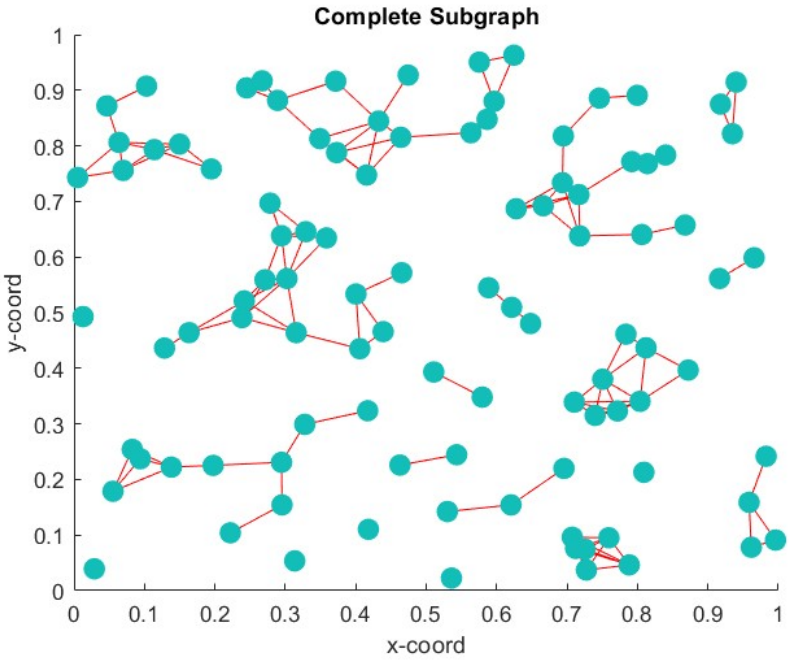

Figure 2: The full graph of routers and their links

increasing η-function. Of course, what is good is subjective. It might not be necessary to find the optimal solution. However, if the optimal solution is to be found, the best performance was found with:

$$\eta(t) = \frac{t^{0.8}}{100}$$

The plot for the potentials evolution can be found in figure 5.

It would be interesting to try more complex η-functions. One example would be to have η proportional to the rate of change in the potential over the last few time-steps, for example. Implementing this strategy proved to be a bit tricky. Finding what it means for the potential to be changing ”rapidly” or ”slowly”, and deciding how much to adjust $\eta(t)$ based on the rate of change in the potential was difficult and would need a more in-depth analysis of the algorithm. This is thus left as an exercise to the reader...

Another way to make the algorithm more efficient would be to pre-process the graph. The isolated vertices has no effect on the potential and can be set arbitrarily, and as the algorithm chooses these to evaluate at times it creates an unnecessary slow-down. Furthermore, the graph could be portioned in the respective connected components. Then one could evaluate each connected component separately. This way, if the algorithm finds a zero-potential colouring on one component, the algorithm can exist and focus on the next component. If it does not find a solution after a certain number of iteration it exits. On the small connected component it exited after a solution with potential 3 was found, while it had a max number of iterations of 800 (or reaching zero-potential). Here a $\eta = t^{1.5}/100$ was used, with t resetting on each new component. The expected number of iterations before optimality was found decreased substantially, to about 1000. However, optimality cannot be guaranteed now, as because of the max number of iterations limit and the randomness of the graph (not that optimality ever can be guaranteed in finite time on a single run with a random algorithm).

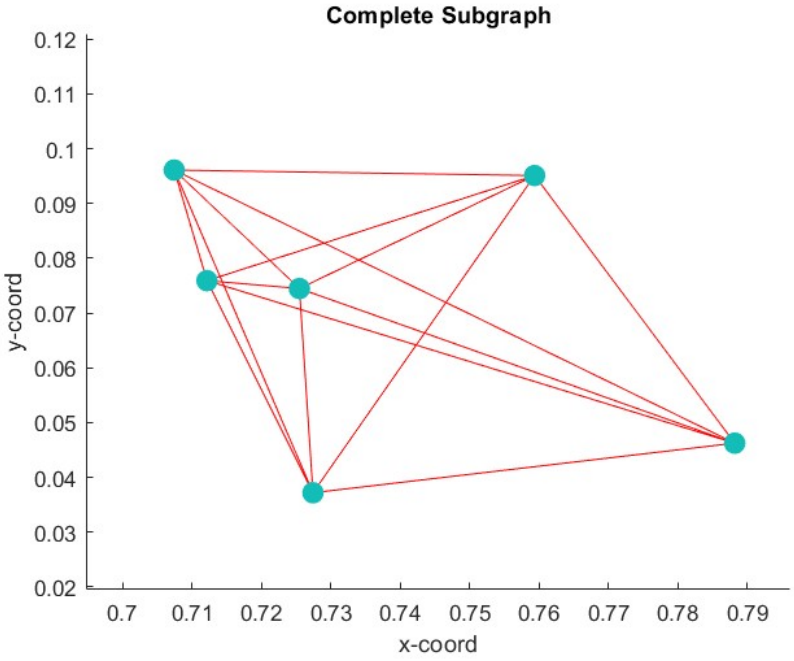

Figure 3: A complete subgraph in the network.

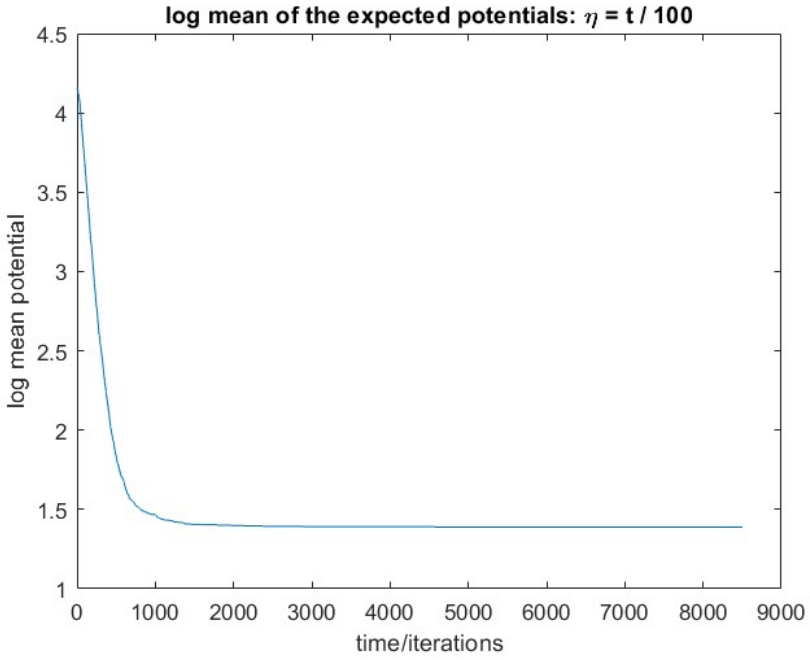

Figure 4: Log average potential between 100 runs to optimality with linear inverse noise / temperature function.

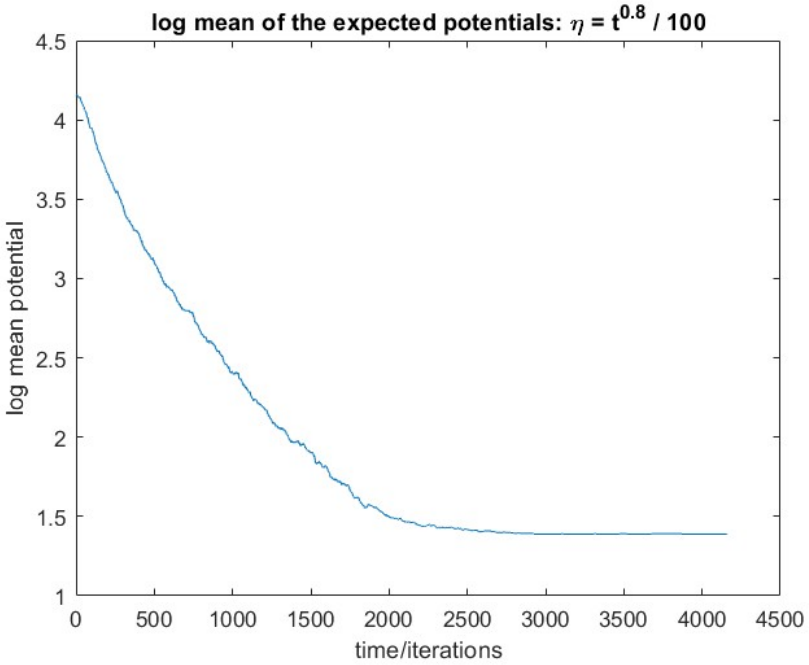

Figure 5: Log average potential between 100 runs to optimality with a slower than linear inverse noise / temperature function.