Lab5 - Dynamic processes on graph

Martina Martini s306163
Politecnico di Torino

Assignment Write a piece of code to simulate dynamic processes on graphs. The simulator should be able to:

i) generate in an efficient way a either G(n,p) graphs or regular grids (with order of 100k nodes);

ii) handle in an efficient way the FES (resort on properties of Poisson processes).

Deliver the code along with a brief report, in which you clearly describe:

i) the data structure you use;

ii) which are the events and how the FES is handled.

ii) the algorithm according to which you generate samples of G(n,p) graphs.

Furthermore, for n=100k, p= 10^-4 compare the empirical distribution of the degree with analytical predictions. Build a q-q plot and execute a χ^2 test.

1 Random elements, inputs and main assumptions

1.1 Introduction

The G(n, p) graph is a typology of graphs that allows us to handle dynamical processes. This type of graph is the result of a random construction process, whose parameters are the number of nodes n and the probability of adding an edge p or not adding it *1 - p*. Moreover, the degree of each node (i.e., the number of edges) follows a binomial distribution. Furthermore, working with such a graph can be challenging because of its quadratic computational complexity. However, with this simulation, I am going to show how to generate a G(n, p) graph intuitively - without any help of predefined functions - and how the Future Event Set helps us in a dynamic context. Finally, I will show and comment on the comparison between the empirical and the analytical distribution, the q-q plot and the χ^2 test results.

1.2 Stochastic elements

To simulate such a dynamic process on a graph, some stochastic elements must be considered. Firstly, the interarrival times for scheduling the events when the FES is working are exponentially distributed with parameter $\frac{1}{\lambda}$. Secondly, the choice of the state variable is random between +1 and -1 for each node at the first stage. Thirdly, the number of edges k is Poisson distributed as follows: $k \sim Poisson(np)$. This result derives from the probability that the degrees (i.e., the number of edges) of the node v equal to k is $\binom{n-1}{k} p^k (1-p)^{n-1-k}$, so we can say that

$$Pr(deg(v) = k) \sim \frac{(np)^k e^{-np}}{k!}.$$

Moreover, the choice of the k nodes linked with each node (i.e., the neighbors) is randomly executed. Finally, a high seed is introduced for reproducibility purposes.

1.3 Input parameters

For what concerns the input parameters, I assume to set the following fixed variables: the number of nodes n firstly equal to 100_000 and the to 10_000 (see the following paragraph), the probability of adding an edge p equal to 0.001, the end time of the simulation to 1000, a high seed to 1322 and, finally, some arrival rates as $lambdas = [0.2, 0.6, 1.2, 2]$.

1.4 Main assumption

Regarding the assumption I made, just one main specification needs to be explained for the simulation purposes. After working with 100_000 nodes, I saw that the graph takes a small amount of time to be created, but the update process (see the following paragraph) took always more than two hours. For this reason, I lowered the number of nodes to 10_000, seeing similar results, so that the following considerations and analyses are done under this assumption.

2 Output metrics, data structures and further analyses

2.1 Methodology and data structures

For each arrival rate λ presented above, the same system structure is applied in order to compare it at the end of the simulation. Generally, at the first stage, nodes and graphs are created, then, the FES is initialized and, finally, the simulation starts updating the state variables by means of the FES. Going into the details, the pseudo-codes are presented below:

2.1.1 create_nodes(n) function

This function generates n instances of the class Node. This class is characterized by the index, the vote and the list of neighbors (i.e., the linked nodes). Furthermore, since the voter model is applied, the state variable representing the vote (+1 or -1) is chosen randomly and associated with each node.

2.1.2 create_graph(n, p, nodes) function

This function handles each node of the graph generating its neighbors: the graph is structured as a dictionary whose keys are the nodes and whose values are the linked nodes.

Basically, the loop works for each node as follows:

step 1 choose k number of edges Poisson distributed

step 2 store k as degrees of the node

step 3 select all the other nodes except for the one taken under consideration

step 4 for each iteration until k is reached, choose a random node from the other and add it as a neighbor

step 5 once the first version of the graph is created, check if the node under consideration is present in the list of neighbors of some other nodes: if the node i is present in the list of the node j, then add j to the neighbors of i

step 6 finally, remove the duplicates in the neighbors list

step 7 return the dictionary-structured graph and degrees

2.1.3 initialize_fes(lambda_rate, nodes)

This function takes as parameters the arrival rate and the list of nodes, then it initializes the FES as a priority queue (FIFO type). Each event of type 'update' is associated with a single node and a single arrival time. Each arrival time is defined by the exponential distribution with parameter lambda_rate.

2.1.4 simulate(graph, fes, end_time) function

This function handles the events in the FES and progressively updates the graph. In fact, until the FES is full and the simulation time is not expired, the system processes the events of the type 'update' one by one. For each event, the system takes the associated node and applies the so-called Voter Model. It consists of a model in which all the nodes are waken up at the same rate: when a node is awake, its vote (i.e., state variable) changes and becomes equal to its neighborhood. In our simulation, if the current node has vote equal to -1, then all its neighbors change their vote - if different - copying the vote of the current node. the same works in the case of +1. After that, the next 'update' event is scheduled taking into consideration one of the neighbors.

Progressively, the system updates the neighbors and the change the state variables hoping to the reach the consensus state, in which every node has the same associated vote.

2.2 Output metrics and analyses

Firstly, the number of nodes without any edge and the number of unreached nodes (the ones whose vote has not changed) are displayed. Looking at the results, for non of the rates there are non-linked or unreached nodes.

Secondly, the output metrics used to evaluate the results of the simulation consist of what is shown in Figure 1. In the first graph, we are evaluating the number of edges for each node, while in the second one the comparison between analytical and empirical degrees distribution. As we can notice, for each arrival rate the degrees are well distributed in a bell shape.

Finally, I built the q-q plot and I computed the χ^2 test, as shown in Figure 2. Unfortunately, the points in the graph are not lying as expected on the bisector: this means that some parameters must be tuned to find the best.

Instead, the p-value reaches the highest value for rates less than 1, but it does not give satisfaction in the case of higher rates.

For clearer evaluations, please refer to the code provided.

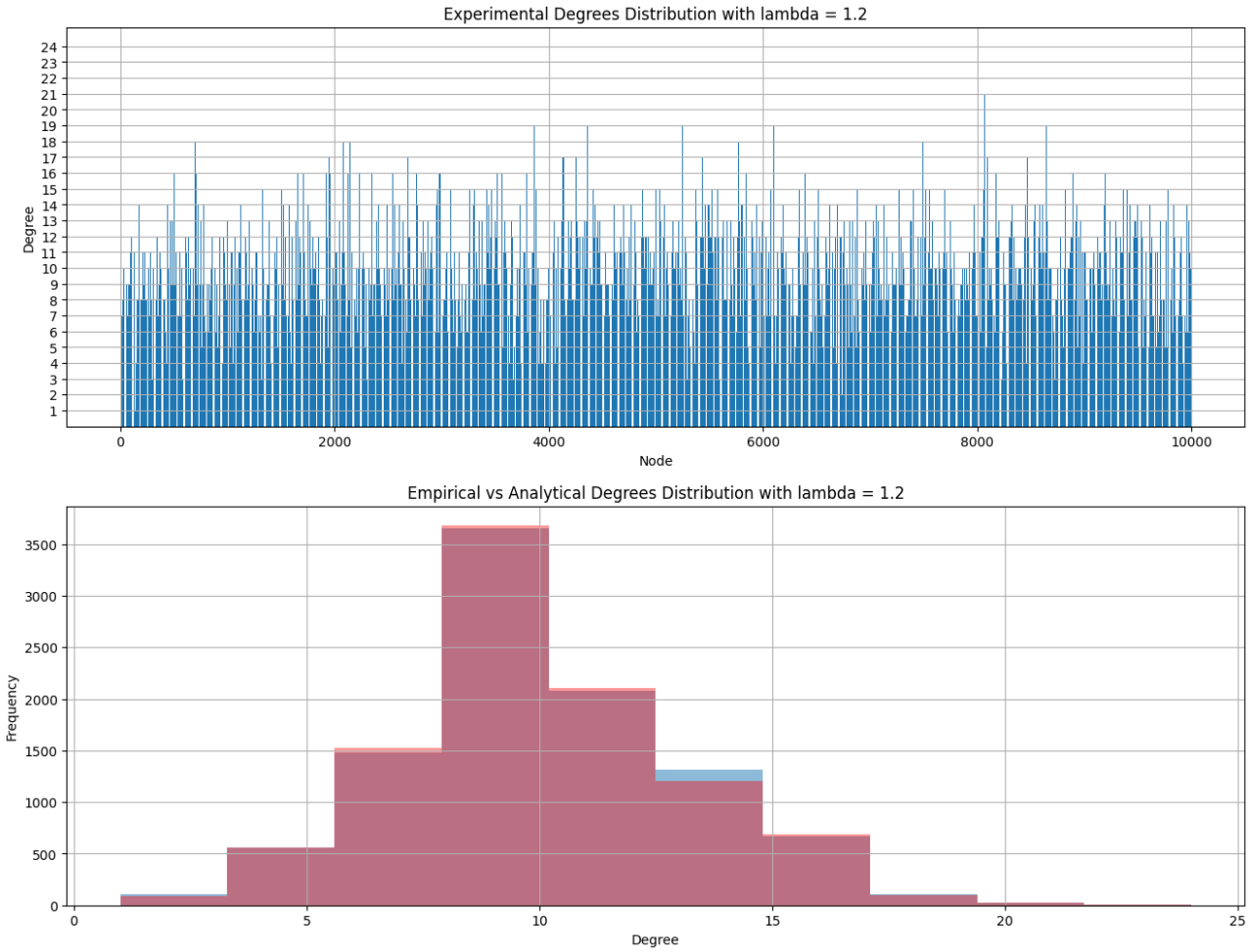

Figure 1: Main output metrics

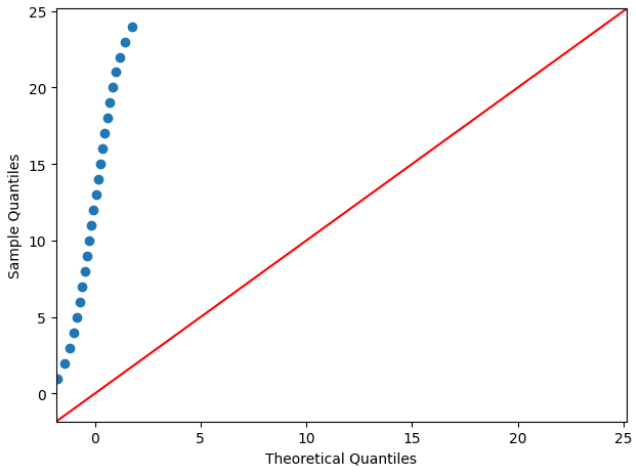

Figure 2: Main output metrics