Lab L5: Dynamic processes on graphs

Emanuele Pietropaolo
Politecnico di Torino
Student id: s319501
emanuele.pietropaolo@studenti.polito.it

***Abstract*—Generating a G(n,p) graph efficiently can be a challenging task. Building an Erdős-Rényi model involves performing a Bernoulli experiment for each pair of nodes. This paper presents some methods to efficiently generate such graphs and to perform dynamic processes on them. The computational correctness is evaluated by visualisation (QQ-plot) and χ^2 test.**

I. Problem overview

Random graphs are a special type of graph that can be described by a probability distribution, or random process, that generates them. Their number of edges depends on a stochastic process starting from a given number of nodes. The process involves connecting random nodes with an edge based on probability. The most important and widely used model describing this type of graph is called the **Erdős-Rényi model**.

II. Proposed approach

A. Algorithm to generate samples of G(n,p) graphs

This report examines two methods of generating samples of random graphs:

- **Generating directly an Erdős–Rényi graph**: this method involves iterating over all the nodes and for each one iterating over all the nodes not already seen. This method then create an edge between a pair of nodes based on the *edge probability*. This algorithm has complexity $O(n^2)$ but it tends to $O(nlog(n))$ because the second for loop is decreasing over time.

Algorithm 1 Erdős–Rényi method

```
Input: n = number of nodes, p = edge probability
for i in range(n) do
  for j in range(i + 1, n) do
    if random() < p then
      create edge(i,j)
    end if
  end for
end for
```

- **Take advantage in case of small edge probability**: when p can be considered small in front of the number of nodes, it can be calculated the number of expected edges (*m*) that the graph will have through the formula:
$$\frac{n*(n-1)*p}{2}$$
Then create a for loop that for each edge picks two random nodes and connects them. This algorithm has complexity $O(m)$ that for small values of p tends to $O(n)$.

Algorithm 2 Taking advantage of small edge probability

```
Input: n = number of nodes, p = edge probability
m = n * (n − 1) * p/2
for i in range(m) do
  while True do
    i,j = generate two nodes index
    if i == j or jinnode[i][neighbors] then
      continue
    else
      createedge(i, j)
      break
    end if
  end while
end for
```

B. Data structure

I used a **dictionary** to handle the graph. Each key of the dict is an index of a node and each node is a nested dictionary with a list of neighbors and a state variable.

C. How the FES is handled

The FES handle the wake-up events and at the beginning is filled up with only one event, the wake-up for a random node. Then after each wake-up event, it's filled with a new of such events associated with a new random node. The wake-up process is supposed to follow a Poisson process so the inter-times are generated from a exponential distribution with $\lambda = 1$

I modelled the **Voter Model** so for each event is updated the state variable of the waked up node (v) with the value of a random neighbor (w), following the formula:

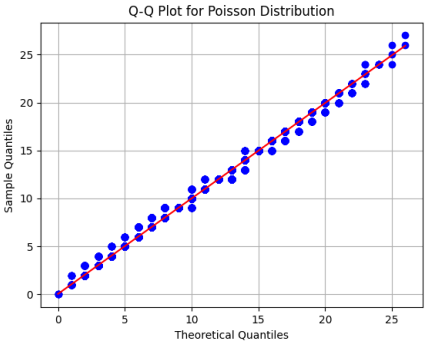
$$x_v(t_v^+) = x_w(t_v^-)$$

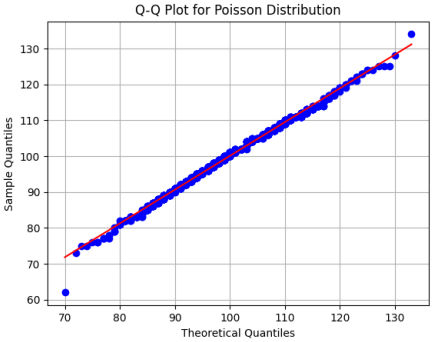
III. Results

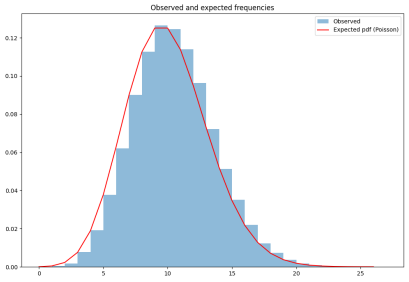
My algorithm is capable of handling random graphs with the order of $100k$ nodes. In Fig. III is the Q-Q plot for the degree distribution of the graph with $100k$ nodes and an edge probability of $p = 10^{-4}$

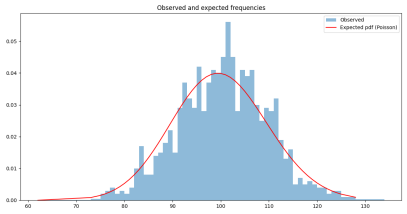
In Fig. III is the expected and the observed degree distribution of the graph with $100k$ nodes and an edge probability of $p = 10^{-4}$. The χ^2 test return a p value = 1.

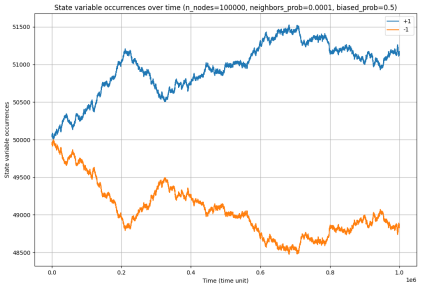
We can also see the behaviors in Fig. III and III where in the first is hard to have a convergence because of the few edges between nodes, and in the second we can see this stable flow

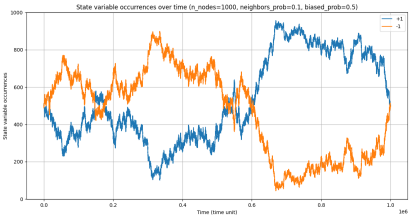


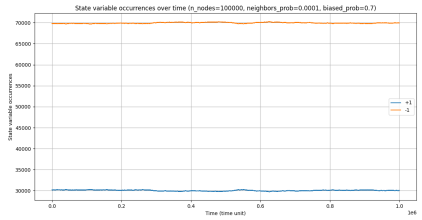


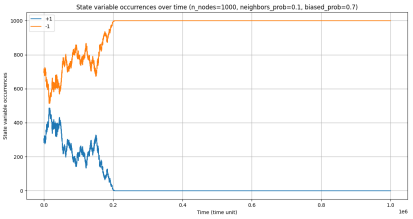












caused by the probable number of subgraphs that is caused by the small value of edge probability.

In this second sets of graph we can see a smaller graph (with 1000 nodes and an edge probability of 0.1).

In Fig. III and III we can see that in the second case, it is more likely that the graph will reach a stable state starting from a biased condition.

IV. Conclusion

In conclusion, the proposed approach demonstrated that can successfully generate random graphs.