IIA SF5 Week1 Interim Report 1

htl28

May 2024

1 Introduction

The simplest model for a network is a random graph, denoted $G(n, p)$. The network consists of n nodes, each pair of nodes is connected with probability p.

We investigate two methods of sampling a random graph. The naive method loops through all possible pairs of edges and generates an edge with probability p, with the number of edge in the graph being the random variable. The two-stage method first computes the probability mass function of the number of edges in the graph, m, then generates a value m before adding edges uniformly to the network. We also observe how the degree distribution is affected by p. Finally, we study components size in the network as a function of p.

1.1 Number of edges in a network as a function of p

Denote number of edges as m, and probability of an edge by p. By generating a sequence of graphs with various p values, we plot histograms to observe the sampled distribution of the random variable m for a 1000 runs. Then we calculate the probability mass function of m for various values of p, while keeping the number of nodes, $n = 100$, constant.

p	Theoretical mean	Theoretical Var	Sample Mean	Sample Var
0.1	495	445.5	494.51	396.51
0.3	1485	1039.5	1486.02	1053.66
0.5	2475	1237.5	2476.08	1161.90
0.7	3465	1039.5	3462.69	1058.74
0.9	4455	445.5	4454.70	456.17

Table 1: Mean and variance of m with n=100

The sample distribution, mean and variance closely resembles those of the theoretical values.

1.2 Degree distribution as function of p

Denote the degree of a node as k. The probability of a node having exactly k edges is:

$$P(k) = \binom{n-1}{k} * p^k * (1-p)^{n-1-k} \tag{1}$$

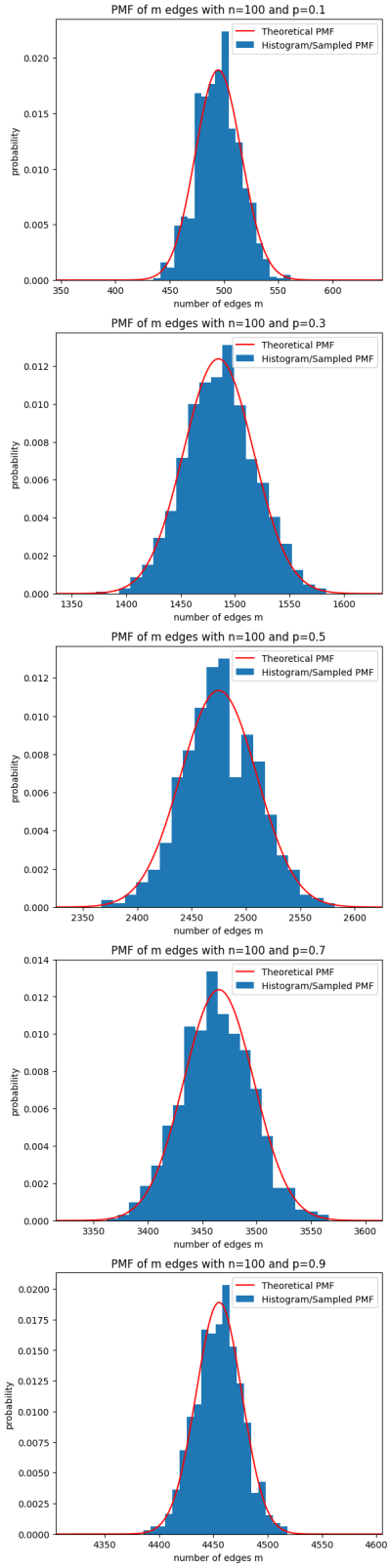

Figure 1: PMF of m, n=100

p	mean	variance
0.1	9.90	8.91
0.3	29.70	20.79
0.5	49.50	24.75
0.7	69.30	20.79
0.9	89.10	8.91

Table 2: Mean and variance of degree distribution with n=100

Now, let

$$p = \frac{\lambda}{n-1}$$

for some constant λ. Using equation 1 and substituting for p, we calculate $P(k)$ to be

(2)

$$\begin{aligned} P(k) &= \binom{n-1}{k} * \frac{\lambda}{n-1}^{k} * (1 - \frac{\lambda}{n-1})^{n-1-k} \\ &= \frac{\lambda^k}{k!} * \frac{(n-1)(n-2)...(n-1-k)}{(n-1)(n-1)...(n-1)} * (1 - \frac{\lambda}{n-1})^{n-1-k} \\ n \xrightarrow{=} \infty & \frac{\lambda^k}{k!} e^{-\lambda} \end{aligned}$$

As n approaches infinity, $P(k)$ has a poisson distribution. We observe that the mean is always equal to λ and as n increases, the variance asymptotically approaches λ too, which are properties of the poisson distribution.

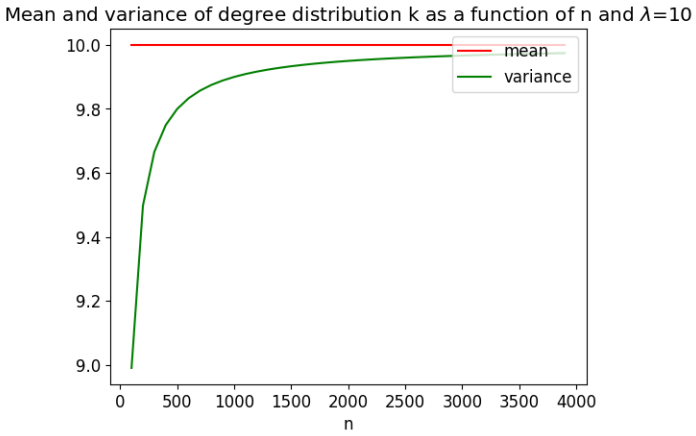

Figure 2: Mean and variance of k with $\lambda = 10$

1.3 Time complexity of graph sampling using the two-stage method

The naive method of generating $G(n,p)$ loops through all $\binom{n}{2}$ possible edges. We attempt to sample a graph $G(n,p)$ using a different two-stage method. First, pick m according to the computed PMF, then randomly add m edges to the network. For various large numbers of n, we record the average time taken to sample the graphs using the two different algorithms and plot a log-log plot.

As expected, the naive method has a time complexity of $O(n^2)$ and the two stage method has a time complexity of $\approx O(n)$.

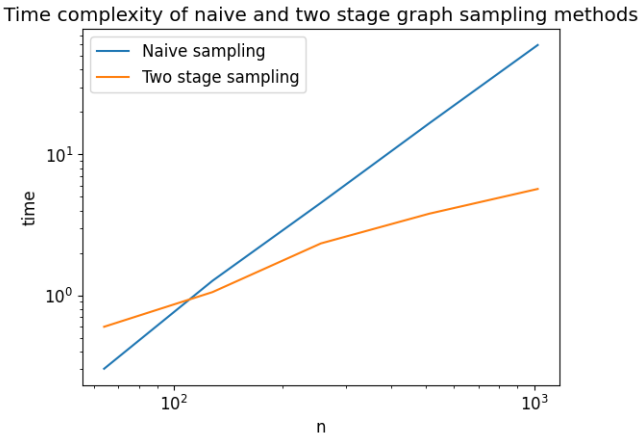

Figure 3: Time complexity of two different graph sampling algorithms

1.4 Component size as a function of p

A network is in general fragmented into multiple components. Randomly picking a node, we use breadth first search to see how component size changes as p varies from 0 to 0.001, by averaging over 50 runs.

At approximately $p = \frac{1}{n-1} = \frac{1}{4095}$, we indeed observe a change in component size to be much larger (as a fraction of total number of nodes).

We can think of it this way. [1] For a random graph $G(n,p)$ where $p = \frac{\lambda}{n-1}$, we know that λ gives the mean degree, so starting from a random vertex it will have λ neighbours on average, and its neighbors will have λ^2 neighbours. So after s

steps we would have λ^s vertices within the distance s from the initial vertex. If $\lambda > 1$ this number will grow exponential and hence most of the nodes have to be connected into the giant component. We define the giant component as the component of $G(n,p)$ whose order is $O(n)$.

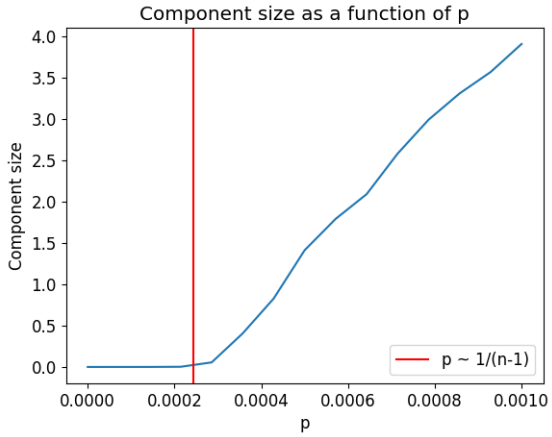

Figure 4: Component size as a function of p, n=4096

2 Appendix

Python Code for two-stage algorithm

```
class Network():
    def __init__(self,num_nodes):
        self.adj = {i:set() for i in range(num_nodes)}
        self.num_edge = 0

    def add_edge(self,i,j):
        self.adj[i].add(j)
        self.adj[j].add(i)
        self.num_edge+=1

    def neighbors (self,i):
        return self.adj[i]

    def edge_list(self):
        return [(i,j) for i in self.adj for j in self.adj[i] if i<j]

class Two_Stage_Network(Network):

    def __init__(self, num_nodes,prob):
        super().__init__(num_nodes)

        #stage 1
        total_edge = math.comb(num_nodes,2)
        m = np.random.binomial(total_edge,prob)

        #stage 2
        while self.num_edge!=m:

            i, j = np.random.randint(0, num_nodes,2)

            if i!=j and j not in self.neighbors(i):
```

```
            self.add_edge(i,j)
```

References

[1] North Dakota State University, Random Graph Notes, Chapter 2.