IIA SF5 Week 2 Interim Report II

htl28

May 2024

1 Introduction

In a network, the degree k represents the number of edges connected to a node. The excess degree κ represents the number of edges connected to a neighbour of the node we are inspecting. This week, we generate networks that (approximately) follow a given degree distribution, known as a *configuration model* . Then we explore the *friendship paradox*, a mathematical fact stating that on average, the mean degree is less than the mean excess degree. Finally, we investigate how component size in the configuration model changes with the mean excess degree. We also briefly investigate the effect of degree correlation on expected degree and expected excess degree.

2 Assignments

2.1 Generating configuration models with Poisson and Geometric Degree Distribution

The python code generating the graphs can be found in the Appendix. For the Poisson distribution, the parameter $\lambda =< k >$. For the geometric distribution, the parameter $p = \frac{1}{<k>+1}$. $< k >$ is the mean degree.

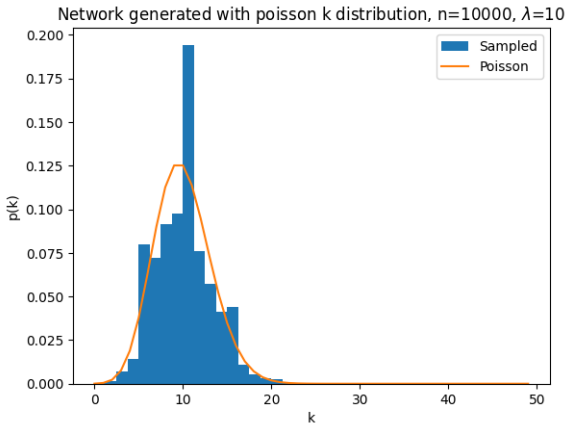

Figure 1: Poisson Distribution Configuration Network with 10,000 nodes and mean degree 10

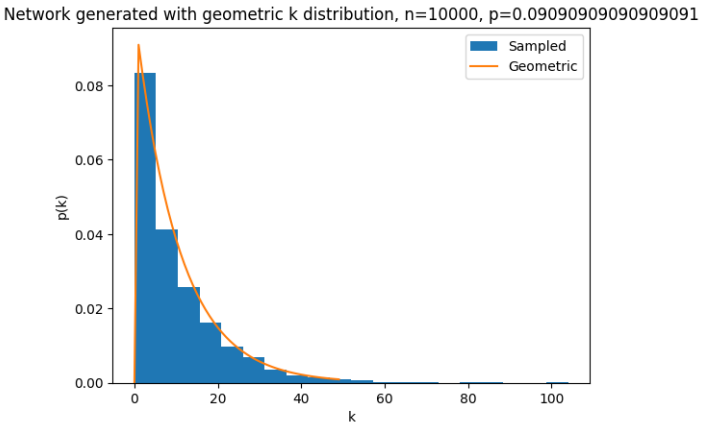

Figure 2: Geometric Distribution Configuration Network with 10,000 nodes and mean degree 10

As observed, the degree distributions closely matches the Poisson/geometric distribution PMF.

2.2 The Friendship Paradox

Given the PMF of degree k, p_k, the PMF of excess degree κ, q_k, follows a different distribution. The Friendship Paradox states that on average, your neighbours will have a greater degree than you. Thus, reusing the previous networks, we sampled a random node and one of its random neighbours and then record both their degrees repeatedly to obtain figure 3 and figure 4.

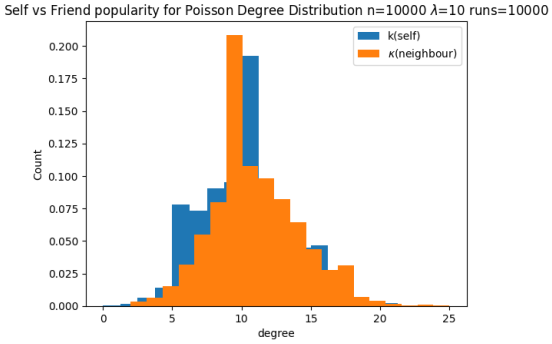

Figure 3: Expected degree = 10.05, expected excess degree = 11.01

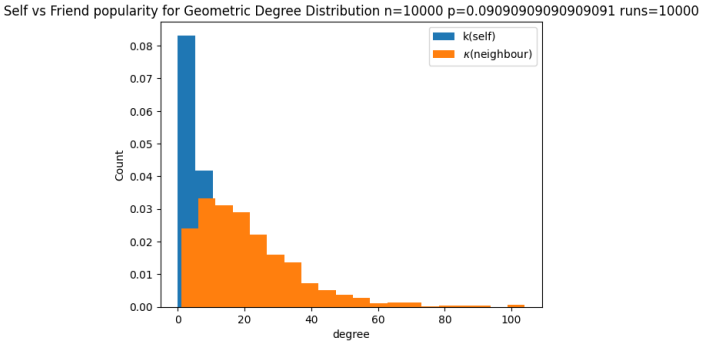

Figure 4: Expected degree = 10.10, expected excess degree = 21.46

We observe for both distributions that the mean excess degree $< \kappa >$ is more than the mean degree $< k >$. To further verify the friendship paradox, we repeatedly sample a random node, obtain it's degree k_i and calculate

it's mean excess degree κ_i (instead of the degree of just one of its neighbours). We calculate $\Delta_i = \kappa_i - k_i$ to obtain figure 5 and figure 6. If a node of $k = 0$ is sampled, it contributes to the calculation of $< k >$ and $\Delta_i = 0$, but only nodes of $k > 0$ contribute to calculating $< \kappa >$.

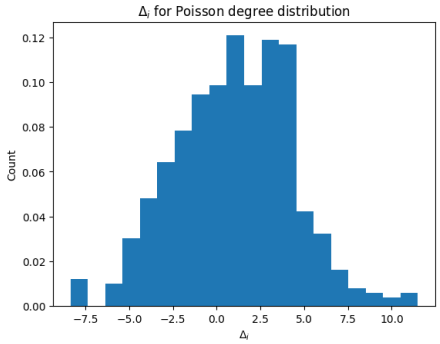

Figure 5: $< \Delta_i > = 0.98$

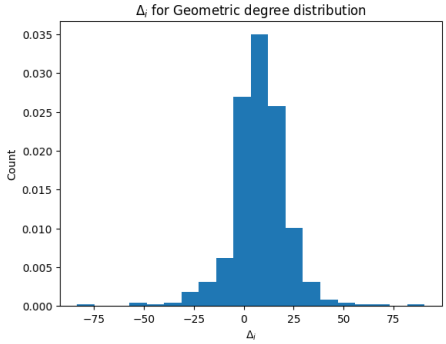

Figure 6: $< \Delta_i > = 8.06$

The means are indeed greater than zero.

2.3 Excess degree distribution q_k for the configuration model

Assume that we randomly pick a node i with $k_i > 0$ in a network with n nodes. What is the probability of randomly choosing a neighbour with $\kappa_i = k$? We first note that a network has $2m = \sum_i k_i$ half-edges. For a neighbour of degree k, there are k half-edges along which we can approach this neighbour, and the total number of half-edges (excluding the edge followed to get to the random node) is $2m - 1$. The total number of such neighbour with degree k in the network is $p_k * n$. Therefore in the limit of large n :

$$q_k = \frac{k * n * p_k}{2m - 1} \approx \frac{k * n * p_k}{2m} = \frac{k p_k}{\sum_{k'} k' p_{k'}} \tag{1}$$

We verify equation 1 by sampling a few configuration graphs of Poisson and Geometric degree distribution.

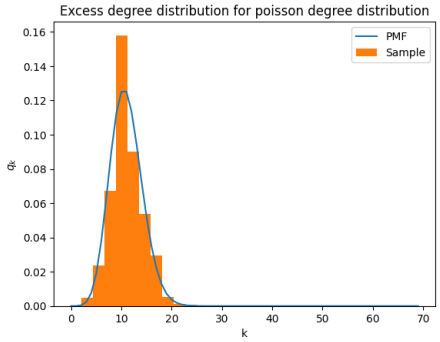

Figure 7: Sampled and exact q_k for Poisson

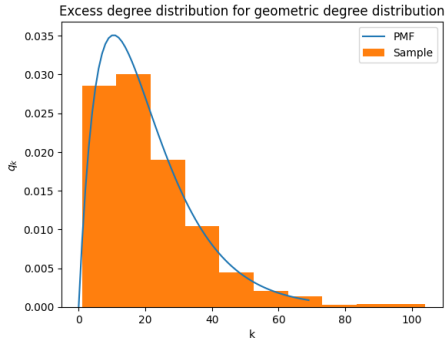

Figure 8: Sampled and exact q_k for geometric

We now sample graphs for a range of λ and p to investigate how the expected degree and expected excess degree changes with the variance of the degree distribution.

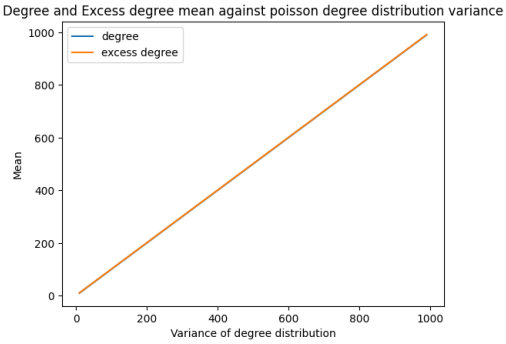

Figure 9:

For the Poisson degree distribution, we know expected degree = variance = λ. Here we observe expected excess degree $\approx$ expected degree, but in truth we will prove below that $< \kappa >=< k > +1$.

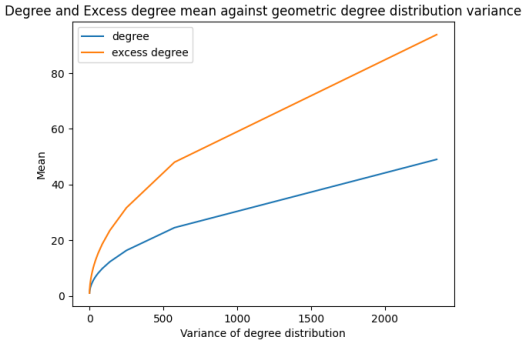

Figure 10:

For the Geometric distribution, we observe the expected degree to always be less than or equal to the expected excess degree. Both quantities increase as the variance increases, however, the relationship is non-linear.

2.4 Generating function for the excess degree distribution

The generating function for distribution p_k is

$$G(z) = \sum_k z^k p_k \tag{2}$$

Differenting with respect to z, we obtain

$$G'(z) = \sum_k k z^{k-1} p_k \tag{3}$$

Letting z=1,

$$G'(1) = \sum_k k p_k \tag{4}$$

and

$$G_1(z) = \frac{zG'(z)}{G'(1)} = \frac{\sum_k zkz^{k-1}p_k}{\sum_{k'} k'p_{k'}} = \sum_k z^k q^k \tag{5}$$

which is the generating function for q_k.
For the Poisson distribution, the explicit expression for the excess degree's generating function is

$$G_1(z) = ze^{\lambda(z-1)} \tag{6}$$

For the Geometric distribution, the explicit expression for the excess degree's generating function is

$$G_1(z) = \frac{zp^2}{(1-qz)^2} \tag{7}$$

where q=1-p.
We can derive the expression for mean excess degree distribution by deriving $G_1'(1)$.
For Poisson :

$$G_1'(1) = \lambda + 1 \tag{8}$$

So the expected excess degree will always be greater than the expected degree.
For Geometric :

$$G_1'(1) = \frac{2-p}{p} \tag{9}$$

Since the expected degree $= \frac{1-p}{p}$. The expected excess degree is always greater than the expected degree, which is shown in section 1.3 too.
Indeed for both distributions, the friendship paradox is true in stating that on average expected excess degree will be greater than expected degree.

2.5 Component size as a function of mean degree $< k >$

Large components in a configuration model start to exist as $\sum_k kq_k > 2$. Using equation 8, we should observe component size starting to grow at $< k >= 1$ for the Poisson model. For the geometric model, using equation 8 component size starts to grow at $< k >= 0.5$

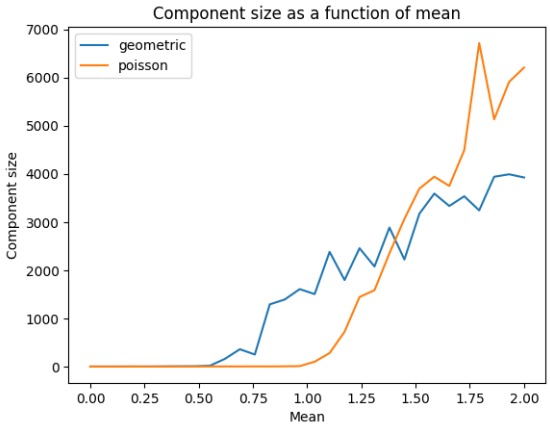

Figure 11:

We observe that component size starts to grow earlier for the geometric distribution compared to the Poisson distribution. The growth for Poisson distribution starts at $< k > = \lambda = 1$ as predicted. The growth for Geometric distribution starts at $< k > = 0.5$, agreeing with the analytical solutions.

2.6 Configuration model with correlated excess degree distribution

To model a configuration model where nodes of high degree are more likely to be connected to nodes of high degree, we sort the nodes by their degree and split them into two halves, the 'popular' half and the 'loner' half. The 'popular' nodes are connected to each other by random and so are the 'loner' nodes. Then using a shuffling function, some nodes between the two pools are swapped to present a degree of randomness.
First we start off with the entirely random, uncorrelated configuration networks. We plot the mean excess degree distribution as a function of k. Then, we overlay the mean excess degree distribution of correlated configuration networks. We expect the mean excess degree to increase with k for the biased case.
The below results have been average overall several sampled networks and large number of nodes to isolate the effects due to degree distribution from effects due to correlation.

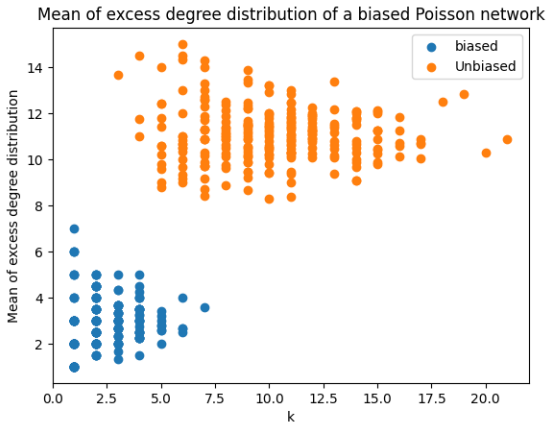

Figure 12: Caption

	Uncorrelated	Correlated
Mean degree	10.12	1.99
Mean excess degree	11.08	2.84
Correlation (k vs $< \kappa >$)	-0.070	0.231

Table 1: Poisson Distribution

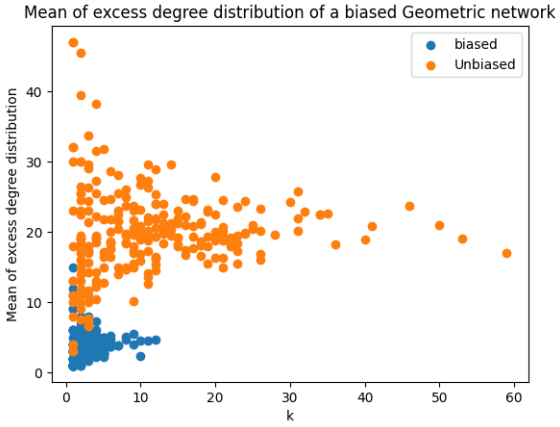

Figure 13: Caption

	Uncorrelated	Correlated
Mean degree	10.91	1.96
Mean excess degree	20.03	3.11
Correlation (k vs $< \kappa >$)	0.04	0.26

Table 2: Geometric Distribution

When correlated, the friendship paradox still holds since $< \kappa >> < k >$ for both distributions. The correlation between node degree and mean excess degree is also positive as expected. However, the mean degrees have decreased significantly.

A Python Code generating Poisson/geometry degree distribution configuration model

```
class Poisson_Configuration_Network(Network):

    def __init__(self, num_nodes,mn=10):
        super().__init__(num_nodes)

        k = np.random.poisson(lam=mn, size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i])])
        S = np.random.permutation(S)

        if len(S)%2:
            S = S[:-1]

        S = S.reshape(-1,2)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)

class Geometric_Configuration_Network(Network):

    def __init__(self, num_nodes, mn=10):
        super().__init__(num_nodes)

        k = np.random.geometric(1/(mn+1), size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i]-1)])
        S = np.random.permutation(S)

        if len(S)%2:
            S = S[:-1]

        S = S.reshape(-1,2)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)
```

The superclass *Network* can be found in the Appendix of week 1's report.

B Python Code for correlated Poisson/geometry degree distribution configuratio model

```
def split(arr):
    freq = Counter(arr)
    sorted_nodes = sorted(freq.items(),key= lambda item:item[1])

    loner = np.array([node for node,f in sorted_nodes[:len(sorted_nodes)//2]
            for _ in range(f)])
    popular = np.array([node for node,f in sorted_nodes[len(sorted_nodes)//2:]
            for _ in range(f)])
```

```
    return popular,loner

def shuffle(arr: np.array, shuffle_factor):
    times = int(shuffle_factor*arr.shape[0])

    for _ in range(times):
        i,j = np.random.randint(0,arr.shape[0],size=2)

        arr[i][0], arr[j][0] = arr[j][0], arr[i][0]

    return arr

class Biased_Poisson_Config_Network(Network):

    def __init__(self, num_nodes,mn,shuffle_factor=0.3):
        super().__init__(num_nodes)

        k = np.random.poisson(lam=mn,size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i])])
        S = np.random.permutation(S)

        popular, loner = split(S)

        if len(popular)%2:
            popular = popular[:-1]

        popular = popular.reshape(-1,2)

        if len(loner)%2:
            loner = loner[:-1]

        loner = loner.reshape(-1,2)

        S = np.concatenate((popular,loner),axis=0)

        S = shuffle(S,shuffle_factor)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)

class Biased_Geometric_Config_Network(Network):

    def __init__(self, num_nodes,mn,shuffle_factor=0.3):
        super().__init__(num_nodes)

        k = np.random.geometric(1/(mn+1), size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i]-1)])
        S = np.random.permutation(S)

        popular, loner = split(S)

        if len(popular)%2:
            popular = popular[:-1]
```

```
popular = popular.reshape(-1,2)

if len(loner)%2:
    loner = loner[:-1]

loner = loner.reshape(-1,2)

S = np.concatenate((popular,loner),axis=0)

S = shuffle(S,shuffle_factor)

for i,j in S:
    if i!=j:
        self.add_edge(i,j)
```