Heidelberg University
Institute of Computer Science
Database Systems Research Group

Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Assignment 6

Degree Correlations and Assortativity

`https://github.com/nilskre/CNA_assignments`

Team Member: Patrick Günther, 3660886,
Applied Computer Science
rh269@stud.uni-heidelberg.de

Team Member: Felix Hausberger, 3661293,
Applied Computer Science
eb260@stud.uni-heidelberg.de

Team Member: Nils Krehl, 3664130,
Applied Computer Science
pu268@stud.uni-heidelberg.de

Problem 6-1 The t-Party Evolving Network Model

At the t-party gender plays no role, hence each newcome is allowed to invite **exactly** one other participant to a dance. However, attractiveness plays a role: More attractive participants are more likely to be invited to a dance by a new participant. The party evolves following these rules:

- Every participant corresponds to a node i and is assigned a time-independent attractiveness coefficient η_i.
- At each time step a new node joins the t-party.
- This new node then invites one already partying node to a dance, establishing a new link with it.
- The new node chooses its dance partner with probability proportional to the potential partner's attractiveness. If there are t nodes already at the party, the probability that node i receives a dance invitation is:

$$\Pi_i = \frac{\eta_i}{\sum_j \eta_j} = \frac{\eta_i}{t\langle\eta\rangle}$$

  where $\langle\eta\rangle$ is the average attractiveness.

1. Derive the time evolution of the node degrees, telling us how many dances a node had.

   The rate at which an existing participant i acquires dance partners is

$$\frac{dk_i}{dt} = 1 - (1 - \Pi(\eta_i)).$$

   Using a series expansion and mean-field approximation we receive for $t \to \infty$

$$\frac{dk_i}{dt} \approx \frac{\eta_i}{\sum_{j=1}^{N-1} \eta_j} = \frac{\eta_i}{t\langle\eta\rangle}.$$

   By integrating

$$dk_i = \frac{\eta_i}{\langle\eta\rangle}\frac{dt}{t} \Rightarrow \int_1^{k_i(t)} dk_i = \frac{\eta_i}{\langle\eta\rangle}\int_{t_i}^{t} \frac{1}{t} dt$$

   we obtain

$$k_i(t, \eta_i) = \frac{\eta_i}{\langle \eta \rangle} \cdot ln(\frac{t}{t_i}) + 1.$$

2. Derive the degree distribution of nodes with attractiveness η.

   To build the cumulative degree distribution function, we first need to derive the time constraint when a participant should have entered the t-party the earliest to have at most k dances.

$$\begin{aligned} k_i(t, \eta_i) = \frac{\eta_i}{\langle \eta \rangle} \cdot ln(\frac{t}{t_i}) + 1 &< k \\ \frac{t}{t_i} &< e^{\frac{(k-1)\langle \eta \rangle}{\eta_i}} \\ t_i &> te^{\frac{(1-k)\langle \eta \rangle}{\eta_i}} \end{aligned}$$

   We know that the number of such participants (here as a continuous number) who had at most k dances is given by $N_{<k} = t - t_i$ (as only one participant joins the t-party at every time step) with t_i being derived from the above time constraint. Thus the cumulative degree distribution function is given by

$$P(k_i < k) = \frac{N_{<k}}{N} = 1 - \int_{\eta_{min}}^{\eta_{max}} p(\eta) e^{\frac{(1-k)\langle \eta \rangle}{\eta}} d\eta.$$

   Note that we needed to average over the fitness/attractiveness distribution and that $N \approx t$ for $t \to \infty$.

   To receive the degree distribution function we take the derivative of the cumulative degree distribution function and get

$$p_k = \frac{dP(k)}{dk} = \int_{\eta_{min}}^{\eta_{max}} p(\eta) \frac{\langle \eta \rangle}{\eta} e^{\frac{(1-k)\langle \eta \rangle}{\eta}} d\eta.$$

3. If half of the nodes have $\eta = 2$, and the other half $\eta = 1$, what is the degree distribution of the network after a sufficiently long time?

$$p_k = 0.375e^{0.75(1-k)} - 0.75e^{1.5(1-k)}.$$

Intuitively, there should only be very small degree nodes in the network as each dancers' fitness becomes neglectible if $t \to \infty$. This can be confirmed in Figure 1.

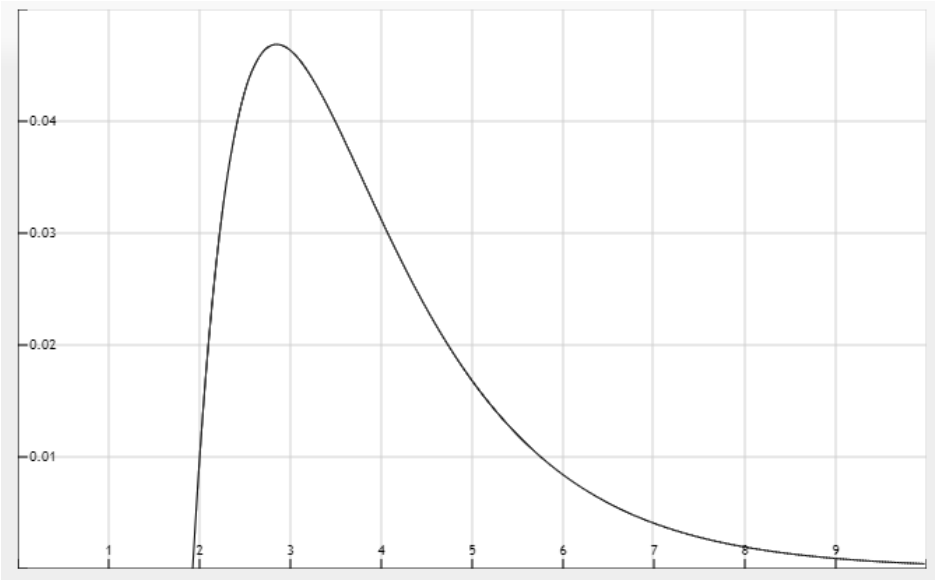


Figure 1: Degree distribution for the given fitness distribution

Problem 6-2 Friendship Paradox Follow-up

December 13, 2021

1 Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Winter Semester 2021/22

1.1 Assignment 6 - Degree Correlations and Assortativity

Students: Felix Hausberger, Nils Krehl, Patrick Günther

[1]:
```
import math
import networkx as nx
import pandas as pd
import numpy as np
from scipy.special import zeta
from scipy import stats
import igraph
import matplotlib.pyplot as plt
```

[2]:
```
# load dataset
raw_data = pd.read_csv('facebook-links.txt', sep="\t", header=None,␣
 ↪names=["source", "target", "timestamp"])
```

[3]:
```
# clean dataset

# 1. remove time-stamp column
data = raw_data.drop(["timestamp"], axis=1)
```

[4]:
```
# 2. Turn the directed network into an undirected network. = done by using␣
 ↪Graph() instead of DiGraph()
# 3. remove multiple edges = done by using Graph() instead of MultiGraph()
G = nx.Graph()
G.add_edges_from(data.itertuples(index=False))
```

2 1. Compute the average number of friends and the average number of "friends of friends" (FOF).

2.1 1.1 Average number of friends

A friend relationship is described by an edge between two nodes. The number of friends of one person (node) is equal to the degree of the node. The average number of friends is then the average degree $< k >$.

[5]:
```
def calc_average_number_of_friends(G):
    node_degrees = np.array(list(G.degree))
    degrees = node_degrees[:,1]
    average_degree = np.sum(degrees) / degrees.shape[0]
    return average_degree, degrees

average_degree, degrees = calc_average_number_of_friends(G)
print("Average number of friends (average degree) = {}".format(average_degree))
```

```
Average number of friends (average degree) = 25.641838351822503
```

2.2 1.2 Average number of "friends of friends" (FOF)

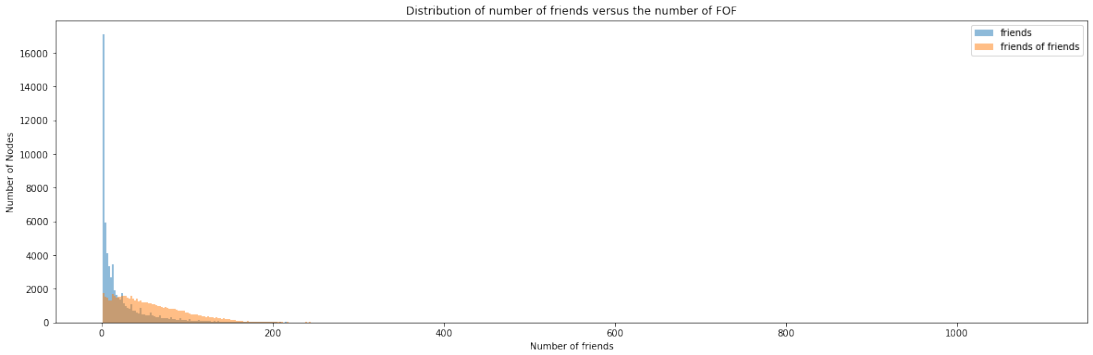
[6]:
```
def calc_average_number_of_friends_of_friends(G):
    number_friends_of_friend_per_node = []

    for node in G.nodes():
        friends = G.neighbors(node)
        friends_of_friend = []
        for friend in friends:
            friends_of_friend.append(G.degree[friend])
        average_number_friends_of_friend = np.sum(np.array(friends_of_friend)) /␣
 ↪len(friends_of_friend)
        number_friends_of_friend_per_node.
 ↪append(average_number_friends_of_friend)

    average_number_friends_of_friend = np.sum(np.
 ↪array(number_friends_of_friend_per_node)) /␣
 ↪len(number_friends_of_friend_per_node)
    return average_number_friends_of_friend, number_friends_of_friend_per_node

average_number_friends_of_friend, number_friends_of_friend_per_node =␣
 ↪calc_average_number_of_friends_of_friends(G)
print("Average number of friends of friends = {}".
 ↪format(average_number_friends_of_friend))
```

```
Average number of friends of friends = 58.3634028072132
```

3 2. Create a plot showing the distribution of number of friends versus the number of FOF.

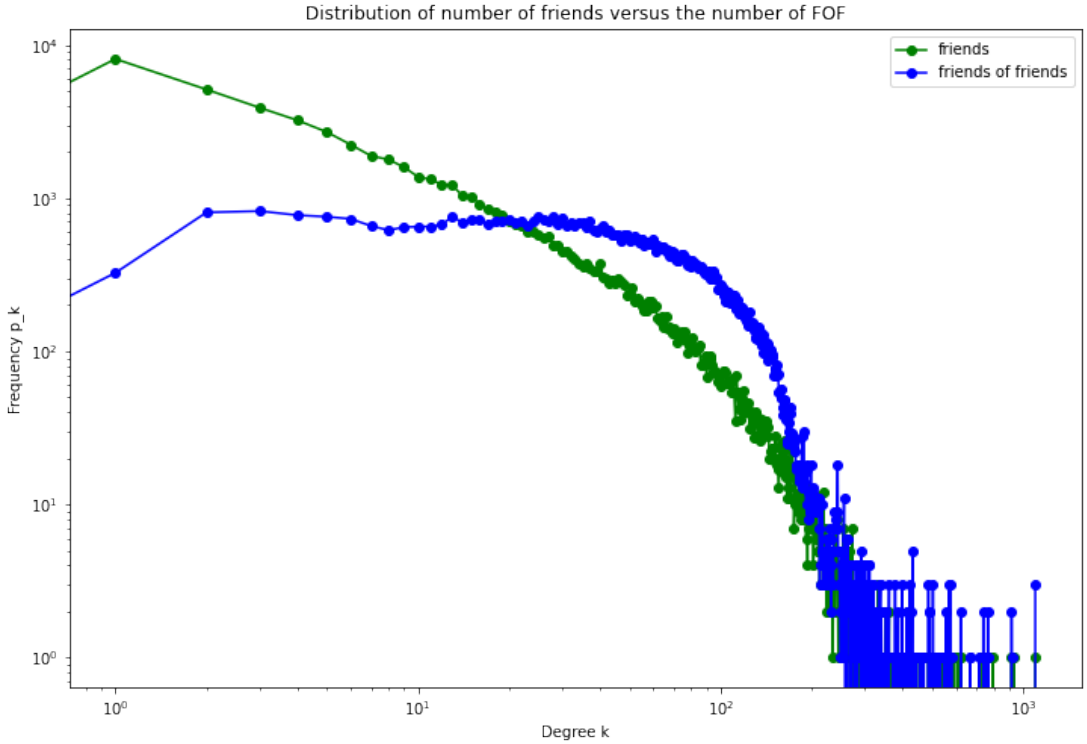
We created two plots for this. The first is a normal histogram without loglog binning. As expected it is not very meaningful. Secondly we created a plot with loglog binning and received a better plot.

[7]:
```
def distribution_friend_fof_normal(degrees, number_friends_of_friend_per_node):
    plt.figure(figsize=(20, 6))
    plt.hist(degrees, 500, alpha=0.5, label='friends')#, density=True)
    plt.hist(number_friends_of_friend_per_node, 500, alpha=0.5, label='friends␣
 ↪of friends')#, density=True)
    plt.legend(loc='upper right')

    plt.title("Distribution of number of friends versus the number of FOF")
    plt.xlabel("Number of friends")
    plt.ylabel("Number of Nodes")
    plt.show()

distribution_friend_fof_normal(degrees, number_friends_of_friend_per_node)
```



[9]:
```
def distribution_friend_fof_loglog(G, number_friends_of_friend_per_node):
    # degree_freq = how many nodes have this (index) number of links
    # number_degrees = degree number from 0 up to the maximum degree
    degree_freq = nx.degree_histogram(G)
    number_friends_of_friend_per_node_freq = np.
 ↪bincount(number_friends_of_friend_per_node, minlength=len(degree_freq))

    number_degrees = range(max(len(degree_freq),␣
 ↪max(number_friends_of_friend_per_node)))

    plt.figure(figsize=(12, 8))
    plt.loglog(number_degrees, degree_freq,'go-', label="friends")
```

```
    plt.loglog(number_degrees, number_friends_of_friend_per_node_freq,'bo-',␣
 ↪label="friends of friends")
    plt.title("Distribution of number of friends versus the number of FOF")
    plt.xlabel('Degree k')
    plt.ylabel('Frequency p_k')
    plt.legend()
    plt.show()

distribution_friend_fof_loglog(G, number_friends_of_friend_per_node)
```



4 3. Repeat the above tasks for a network based on the Barabási-Albert Model with n = 10,000 and m = 7. Discuss and compare the results with that for the Facebook network.

[38]:
```
G_barabasi_albert = nx.barabasi_albert_graph(10000,7) #63731, 13)
```

[39]:
```
print("Real network: Average number of friends (average degree) = {}".
 ↪format(average_degree))
print("Real network: Average number of friends of friends = {}".
 ↪format(average_number_friends_of_friend))
print("Real network: Number of nodes = {}".format(G.number_of_nodes()))
```

```
print("Real network: Number of edges = {}".format(G.number_of_edges()))
#print("Real network: Average clustering = {}".format(nx.average_clustering(G)))
print()
ba_average_degree, ba_degrees = calc_average_number_of_friends(G_barabasi_albert)
print("BA model: Average number of friends (average degree) = {}".
 ↪format(ba_average_degree))
ba_average_number_friends_of_friend, ba_number_friends_of_friend_per_node =␣
 ↪calc_average_number_of_friends_of_friends(G_barabasi_albert)
print("BA model: Average number of friends of friends = {}".
 ↪format(ba_average_number_friends_of_friend))
print("BA model: Number of nodes = {}".format(G_barabasi_albert.
 ↪number_of_nodes()))
print("BA model: Number of edges = {}".format(G_barabasi_albert.
 ↪number_of_edges()))
#print("BA model: Average clustering = {}".format(nx.
 ↪average_clustering(G_barabasi_albert)))
```

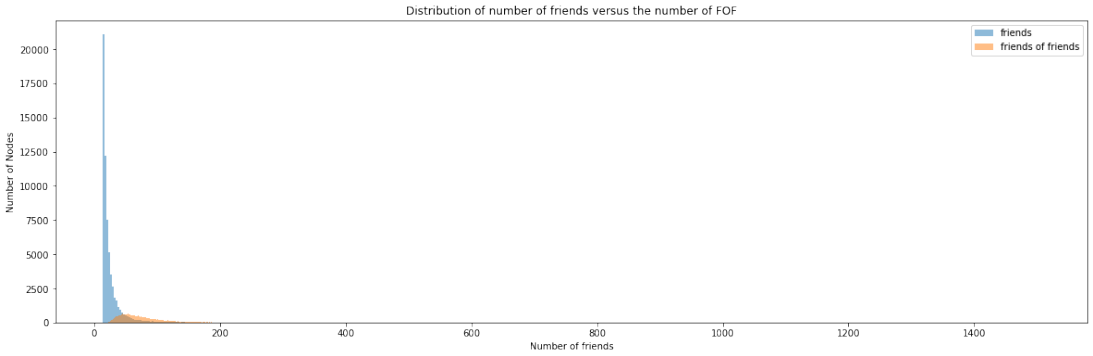
```
Real network: Average number of friends (average degree) = 25.641838351822503
Real network: Average number of friends of friends = 58.3634028072132
Real network: Number of nodes = 63731
Real network: Number of edges = 817090

BA model: Average number of friends (average degree) = 13.9902
BA model: Average number of friends of friends = 37.598432486950315
BA model: Number of nodes = 10000
BA model: Number of edges = 69951
```

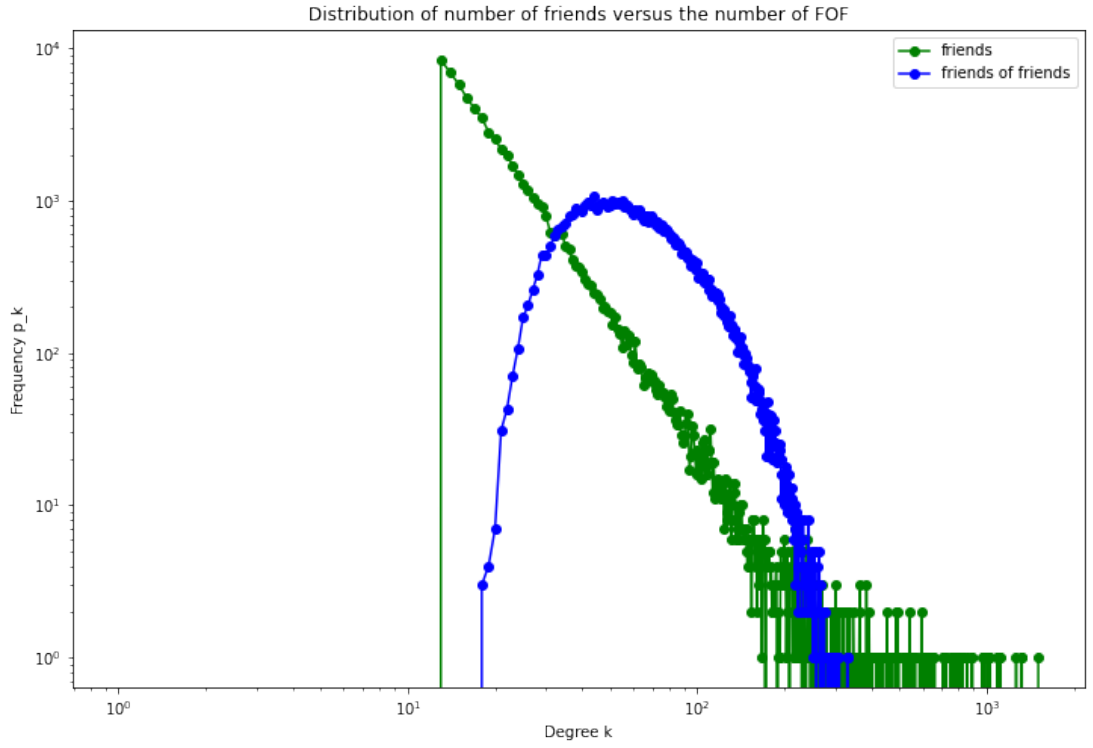
4.1 Comparison and Comment

Both, the average number of friends and the average number of friends of friends is higher in the real network compared to the network generated with the BA model. This follows the fact, that the number of nodes and edges differs between the real network and the BA model (see values above). The network generated with the BA model is much smaller then the real network (the BA model network consists of about $1/6$ of the nodes of the real network and about $7/80$ of the edges of the real network) .

When choosing the BA model parameters $N = 63731$ and $m = 13$, the calculated metrices above get very similar (this does not necessarily imply, that the network topology is similar).

[31]:
```
distribution_friend_fof_normal(ba_degrees, ba_number_friends_of_friend_per_node)
distribution_friend_fof_loglog(G_barabasi_albert,␣
 ↪ba_number_friends_of_friend_per_node)
```





4.2 Comparison and Comment

4.2.1 Real network

- In the real network we observe approximately a powerlaw distribution $p_k \sim k^{-\gamma}$ for the friends curve. The loglog plot shows, that lots of nodes have a small degree (left upper part) and few nodes are hubs and have a high degree (right part at the bottom).
- The FOF curve of the real network is similar to the loglog plot of model B on slide 5-19. This model eliminates growth and keeps preferential attachment.

4.2.2 BA model network

- For the generated network with the BA model, the friends curve follows the typical distibution with the clearly visible powerlaw $p_k \sim k^{-\gamma}$. In comparison to the real network the powerlaw distribution is more accurate. Furthermore the real network contains nodes with a very little degree. The network from the BA model has no nodes with very little degree (which is caused by the parameter m).
- The FOF curve shows that the majority of nodes has a medium degree. Moreover some nodes are hubs (tail of the distribution)

Problem 6-3 Degree Correlation Coefficient

December 13, 2021

1 Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Winter Semester 2021/22

1.1 Assignment 6 - Degree Correlations and Assortativity

Students: Felix Hausberger, Nils Krehl, Patrick Günther

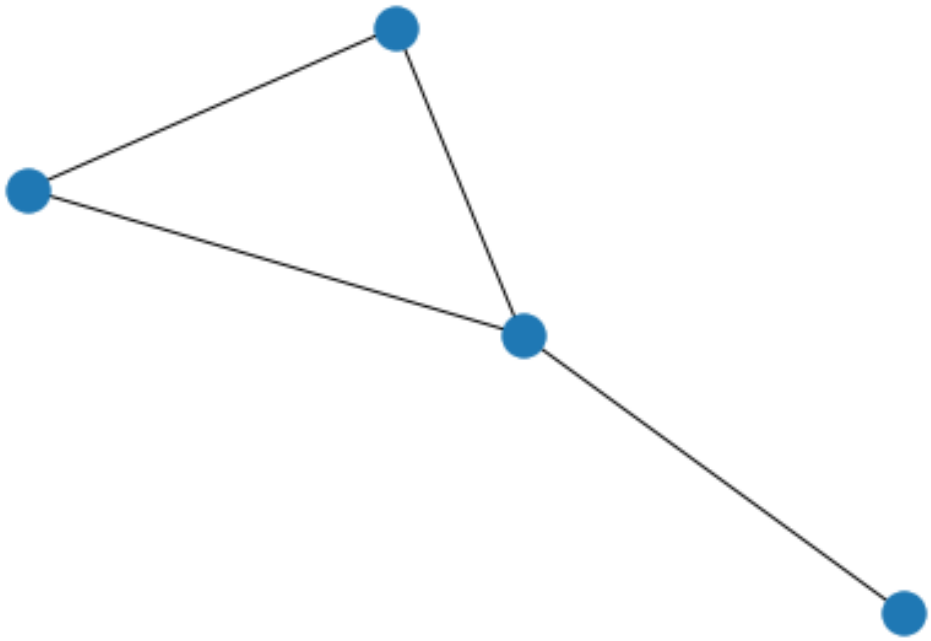
2 1. Compute the degree correlation matrix

[1]:
```
import numpy as np
import networkx as nx
```

[2]:
```
A = np.matrix([[0, 1, 0, 1],
               [1, 0, 1, 1],
               [0, 1, 0, 0],
               [1, 1, 0, 0]])
```

[3]:
```
G = nx.convert_matrix.from_numpy_matrix(A)
```

[4]:
```
nx.draw(G)
```



[5]:
```
G.degree
```

[5]:
```
DegreeView({0: 2, 1: 3, 2: 1, 3: 2})
```

[6]:
```
max_degree = max(deg for n, deg in G.degree)
mapping = {x: x for x in range(max_degree+1)}
deg_corr_mat = nx.degree_mixing_matrix(G, mapping=mapping)
```

[7]:
```
deg_corr_mat
```

[7]:
```
array([[0.   , 0.   , 0.   , 0.   ],
       [0.   , 0.   , 0.   , 0.125],
       [0.   , 0.   , 0.25 , 0.25 ],
       [0.   , 0.125, 0.25 , 0.   ]])
```

(the first column/ row is for degree 0… that could be cut out, since a node with degree 0 never connects to any other node)

3 2. Compute the probabilities q_k of having a degree k node at the end of a random link

[8]:
```
avg_degree = sum(deg for n, deg in G.degree)/len(G.degree)

q_k = {}
for deg in range(max_degree + 1):
    p_k = [deg for n, deg in G.degree].count(deg)/len(G.degree)
    q_k[deg] = (deg * p_k)/avg_degree
```

[9]:
```
q_k
```

[9]:
```
{0: 0.0, 1: 0.125, 2: 0.5, 3: 0.375}
```

4 3. Compute the degree correlation coefficient r

[10]:
```
sigma_squared = sum([(k**2) * q_k[k] for k in q_k]) - sum([k * q_k[k] for k in␣
 ↪q_k])**2

r = []

for j, row in enumerate(deg_corr_mat):
    for k, e_jk in enumerate(row):
        qk = q_k[k]
        qj = q_k[j]
        r.append((j*k*(e_jk-qj*qk))/sigma_squared)

r = sum(r)
```

[11]:
```
print(f"The degree correlation coefficient of the network is {r}.")
```

```
The degree correlation coefficient of the network is -0.7142857142857144.
```

[12]:
```
# to check our computation, we also use the inbuild function of networkx
nx.algorithms.assortativity.degree_assortativity_coefficient(G)
```

[12]:
```
-0.7142857142857143
```

[ ]:
```

```

1. Compute the degree correlation matrix.

   We computed the degree correlation matrix using networkx using inbuild functions (since it is repetitive). The manual process would be to calculate for each cell of the matrix the fraction of edges which connect nodes with the correct degree of i and j. For example, in the graph only one edge connects nodes which have the degree 1 and 3 with each other. There are a total of 4 edges in the graph. Normalized, so that E sums up to 1 (and since e_{ij} as well as e_{ji} are always taken into account) the cells $e_{1,3}$ and $e_{3,1}$ have a value of 0.125. The same process goes for the remaining cells. The degree correlation matrix is shown in Table 1.

	0	**1**	**2**	**3**
0	0	0	0	0
1	0	0	0	0.125
2	0	0	0.25	0.25
3	0	0.125	0.25	0

Table 1: Degree Correlation Matrix

   Manual approach from the tutorial (same results):

   Node degrees: $k = \{2, 3, 1, 2\}$

   Edge list (get it from the adjacency matrix): $\{(1, 2), (1, 4), (2, 3), (2, 4)\}$

   Degree list (insert degree k of the node into the edge list): $\{(2, 3), (2, 2), (3, 2), (3, 1)\}$

   => So two edges exist connecting nodes with degree $(2, 3)$, one edge for $(2, 2)$ (twice due to symmetry), one edge for $(3, 1)$. This is inserted into E and normalized by the sum of all degrees k (8).

$$E = \frac{1}{8} \cdot \begin{array}{c} \begin{array}{ccc} k=1 & k=2 & k=3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 2 \\ 1 & 2 & 0 \end{pmatrix} \end{array} \begin{array}{l} k=1 \\ k=2 \\ k=3 \end{array}$$

2. Compute the probabilities q_k.

   We used formula 7.3 from the lecture slides ($q_k = \frac{k \cdot p_k}{<k>}$). The results are shown in Table 2

3. Compute the degree correlation coefficient r.

   Our solution is $r = -0.714$. We used formula 7.11 from the lecture slides.

   Since $r < 0$, the network is disassortative.

k	q_k
0	0.000
1	$q_k = \frac{1 \cdot 0{,}25}{2} = 0.125$
2	$q_k = \frac{2 \cdot 0{,}5}{2} = 0.500$
3	$q_k = \frac{3 \cdot 0{,}25}{2} = 0.375$

Table 2

Problem 6-4 Degree Correlations in Random Graphs

Consider the random model of undirected random networks $G(N, L)$, where N nodes are connected with L links that are placed uniformly at random. In this model, the probability $p(a_{ij} = 1)$ that an edge exists between nodes i and j is not independent from the existence of other edges since the overall number of edges L in the graph is fixed. Note that this model is distinct from the $G(N, p)$ model in which each edge has an equal probability p of existing.

1. Give an equation for the probability $p(a_{ij} = 1)$ that the edge (i, j) in the graph exists.

   By rearranging formula (3.3) from the lecture slides and solving for p, we obtain

   $$p(a_{ij} = 1) = \frac{L}{\binom{N}{2}} \tag{1}$$

2. Give equations for the probabilities that an other edge (i, j) exists in the graph, conditioned on the existence of the edge (x, y).

   If the edge (x, y) does not exist in the graph L edges are remaining to be distributed between the remaining $\binom{N}{2} - 1$ possibilities for edges to be placed between nodes. If the edge (x, y) does exist there are only $L - 1$ edges remaining. Thus, the probabilities are

   $$p(e_{ij}|e_{x,y} = 0) = \frac{L}{\binom{N}{2} - 1}, \tag{2}$$

   and

   $$p(e_{ij}|e_{x,y} = 1) = \frac{L - 1}{\binom{N}{2} - 1}. \tag{3}$$

3. Derive the ratio of the conditional probabilities to the probability $p(a_{ij} = 1)$.

   Plugging in the formulas we got so far, we get

$$r_0 = \frac{\frac{L}{\binom{N}{2}-1}}{\frac{L}{\binom{N}{2}}} \tag{4}$$

$$= \frac{\binom{N}{2}}{\binom{N}{2}-1} \tag{5}$$

$$= \frac{N(N-1)}{N(N-1)-2} \tag{6}$$

$$= \frac{N^2-N}{N^2-N-2} \tag{7}$$

   and

$$r_1 = \frac{\frac{L-1}{\binom{N}{2}-1}}{\frac{L}{\binom{N}{2}}} \tag{8}$$

$$= \frac{L \cdot \binom{N}{2} - \binom{N}{2}}{L \cdot \binom{N}{2} - L} \tag{9}$$

$$= \frac{\frac{LN(N-1)-N(N-1)}{2}}{\frac{LN(N-1)-2L}{2}} \tag{10}$$

$$= \frac{LN^2 - N^2 - LN + N}{LN^2 - LN - 2L} \tag{11}$$

4. Give the ratios r'_0 and r'_1 for the $G(N,p)$ model.

   In the $G(N,p)$ model, the probability of every edge to be realized is determined by p. Edge probabilities are not dependent on other edges. This means

$$p(a_{ij} = 1) = p(e_{ij}|e_{x,y} = 0) = p(e_{ij}|e_{x,y} = 1), \tag{12}$$

   and thus

$$r'_0 = r'_1 = 1 \tag{13}$$

5. Discuss the implications of using the $G(N, L)$ instead of the $G(N, p)$ model.

   For small networks, in the $G(N, p)$ model, the probability of an edge to exist is the same as in a network with many nodes. In the $G(N, L)$ model, the probability of an edge to exist is higher (if L is kept the same) than in a large network mit many nodes. This is because there are less possible realizations of G, if the number of nodes is small and the probability of an edge to exist is dependent on the existence of other edges in the $G(N, L)$ model. If we look at the ratios r_0 and r_1, we can also see that for L being the same, but N getting small, r_1 decreases faster than r_0 ($-2L$ instead of -2 in the denominator, with the other terms decreasing for N decreasing (does not go to zero)).