Heidelberg University
Institute of Computer Science
Database Systems Research Group

Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Assignment 7

Degree Assortativity and Robustness

`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 7-2 Molloy-Reed Criterion

Consider a configuration model network that has nodes of degree 1, 2, and 3 only, with probabilities p_1, p_2, and p_3, respectively. The degree distribution is given by:

$$p_k = \delta_{k,1}p_1 + \delta_{k,2}p_2 + \delta_{k,3}p_3, \begin{cases} \delta_{k,1} = 3 & \text{if } k = 1 \\ \delta_{k,2} = 2 & \text{if } k = 2 \\ \delta_{k,3} = 2 & \text{if } k = 3 \end{cases} \tag{1}$$

1. Compute the first moment $\langle k \rangle$ and the second moment $\langle k^2 \rangle$ of the degree distribution.

   We assume $\delta_{k,k'}$ to be the dirac-delta-function. For the first and second moment is follows:

$$\langle k \rangle = \sum_{k=1}^{3} k p_k = 1p_1 + 2p_2 + 3p_3 = 3p_1 + 4p_2 + 6p_3$$

$$\langle k^2 \rangle = \sum_{k=1}^{3} k^2 p_k = 1p_1 + 4p_2 + 9p_3 = 3p_1 + 8p_2 + 18p_3$$

   Note that we substitute p_1 with $3p_1$, p_2 with $2p_2$ and p_3 with $2p_3$ as given by equation 1 (slightly confusing by the task description).

2. Using the Molloy-Reed criterion, show that there is a giant component if and only if $p_1 < 3p_3$.

   The Molloy-Reed criteria propagates a giant component exists in case $\kappa = \frac{\langle k^2 \rangle}{\langle k \rangle} > 2$. κ can be calculated as:

$$\kappa = \frac{\langle k^2 \rangle}{\langle k \rangle} = \frac{1p_1 + 4p_2 + 9p_3}{1p_1 + 2p_2 + 3p_3}$$

   which is only true for

$$\begin{aligned} \frac{1p_1 + 4p_2 + 9p_3}{1p_1 + 2p_2 + 3p_3} &> 2 \\ 1p_1 + 4p_2 + 9p_3 &> 2p_1 + 4p_2 + 6p_3 \\ 3p_3 &> p_1 \end{aligned}$$

   Note that we use the LHS declaration of p_k from equation 1.

3. In terms of the structure of the network, discuss the meaning of the condition $p_1 < 3p_3$. Why does the result not depend on p_2?

   For the network to have a giant component, the probability of a node having a single degree should be at most three times as high as the probability of a node having a degree of three. This limits the amount of single degree nodes and promotes a faster growth of a giant component since the average degree will most likely not be close to $\langle k \rangle = 1$, but rather higher (assuming we exclude isolated nodes as in equation 1) since single degree nodes cannot prevail the network.

   The probability p_2 fell apart from the equation shown in subtask 2, leading to the assumption that the emergence of a giant component does not need to be dependent on p_2. This makes sense since we know the constraint $p_1 < 3p_3$ holds, which already leads to the corollar that $\langle k \rangle \geq 1$ and therefore leads to the guaranteed emergence of a giant component.

Problem 7-3 Xalvi-Brunet and Sokolov Algorithm

December 29, 2021

Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Winter Semester 2021/22

Assignment 7 - Assortativity and Robustness

Problem 7-3: Xalvi-Brunet and Sokolov Algorithm

Students: Felix Hausberger, Nils Krehl, Patrick Günther

[1]:
```
import pandas as pd
import networkx as nx
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt
import scipy
```

1. Xalvi-Brunet and Sokolov algorithm

[2]:
```
def xalvi_brunet_sokolov_algorithm(graph, num_iterations, assortative):
    network = graph.copy()
    for i in range(num_iterations):
        links = np.array(list(network.edges))
        degrees = network.degree()
        # choose two random links
        choosen_indices = np.random.choice(range(len(links)), 2, replace=False)
        choosen_links = links[choosen_indices]

        # get corresponding nodes and their node degrees
        corresponding_nodes = choosen_links.flatten()
        corresponding_node_degrees = np.array([degrees[x] for x in␣
↪corresponding_nodes])

        # sort the nodes by their degrees in descending order
        index_array = np.argsort(corresponding_node_degrees)[::-1]
```

```
        ordered_nodes = corresponding_nodes[index_array]
        ordered_nodes_degrees = corresponding_node_degrees[index_array]

        # remove the selected links
        network.remove_edge(choosen_links[0][0], choosen_links[0][1])
        network.remove_edge(choosen_links[1][0], choosen_links[1][1])

        # rewiring
        if assortative == True:
            network.add_edge(ordered_nodes[0], ordered_nodes[1])
            network.add_edge(ordered_nodes[2], ordered_nodes[3])
        else:
            network.add_edge(ordered_nodes[0], ordered_nodes[3])
            network.add_edge(ordered_nodes[1], ordered_nodes[2])

    return network
```

2. Create networks with Xalvi-Brunet and Sokolov algorithm

[3]:
```
df_neutral_network = pd.read_csv('neutral_network.txt', delim_whitespace=True,␣
↪header=None)
```

[4]:
```
G_neutral_network = nx.Graph()
G_neutral_network.add_edges_from(df_neutral_network.itertuples(index=False))
```

[5]:
```
G_assortative = xalvi_brunet_sokolov_algorithm(G_neutral_network, 5000, True)
G_disassortative = xalvi_brunet_sokolov_algorithm(G_neutral_network, 5000, False)
```

[6]:
```
print("Degree Correlation Coefficient")
print("r = 0: neutral network; r < 0: disassortative network; r > 0: assortative␣
↪network \n")
print("Neutral network Degree Correlation Coefficient: {}".format(nx.
↪degree_pearson_correlation_coefficient(G_neutral_network)))
print("Assortative network Degree Correlation Coefficient: {}".format(nx.
↪degree_pearson_correlation_coefficient(G_assortative)))
print("Disassortative network Degree Correlation Coefficient: {}".format(nx.
↪degree_pearson_correlation_coefficient(G_disassortative)))
```

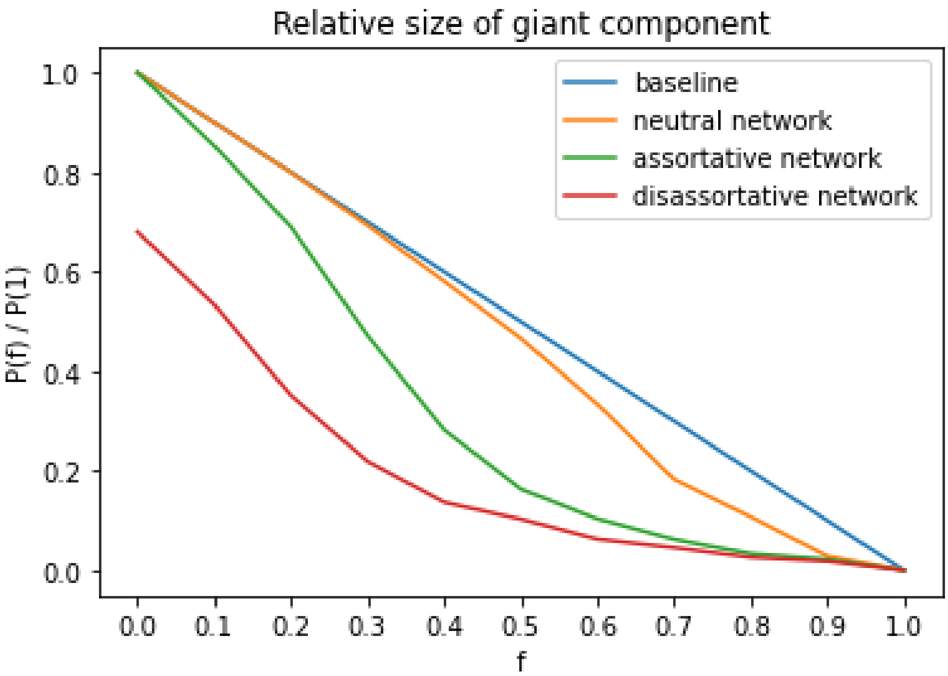
```
Degree Correlation Coefficient
r = 0: neutral network; r < 0: disassortative network; r > 0: assortative
network

Neutral network Degree Correlation Coefficient: -0.009246262701730106
Assortative network Degree Correlation Coefficient: 0.9037799701732246
Disassortative network Degree Correlation Coefficient: -0.6223530885853903
```

3. Plot giant component size

[69]:
```
def get_giant_component_size(network):
    if network.number_of_nodes() > 0:
        giant_component = max(nx.connected_components(network), key=len)
        giant_component_size = len(giant_component)
        return giant_component_size
    else:
        return 0


def get_relative_size_of_giant_component(graph, num_samples=20):
    network = graph.copy()
    number_nodes = network.number_of_nodes()
    f = []

    relative_size_of_giant_component = []
    for f_value in np.arange(0,1.1,0.1):
        giant_component_size = []
        for sample in range(num_samples):
            minimized_network = network.copy()
            number_to_be_removed = int(f_value * number_nodes)

            random_sample = np.random.choice(minimized_network.nodes(),␣
 ↪number_to_be_removed, replace=False)
            minimized_network.remove_nodes_from(random_sample)

            giant_component_size.
 ↪append(get_giant_component_size(minimized_network))
        relative_size_of_giant_component.append(np.mean(np.
 ↪array(giant_component_size)))

    return np.array(relative_size_of_giant_component) / 100

neutral_relative_size_of_giant_component =␣
 ↪get_relative_size_of_giant_component(G_neutral_network)
assortative_relative_size_of_giant_component =␣
 ↪get_relative_size_of_giant_component(G_assortative)
disassortative_relative_size_of_giant_component =␣
 ↪get_relative_size_of_giant_component(G_disassortative)
```

[71]:
```
plt.plot(np.arange(10,-1,-1) / 10, label="baseline")
plt.plot(neutral_relative_size_of_giant_component, label="neutral network")
plt.plot(assortative_relative_size_of_giant_component, label="assortative␣
 ↪network")
```

```
plt.plot(disassortative_relative_size_of_giant_component, label="disassortative␣
    ↪network")
plt.xticks(ticks=range(11), labels=(np.arange(0,11,1) / 10))
plt.legend()
plt.title("Relative size of giant component")
plt.xlabel("f")
plt.ylabel("P(f) / P(1)")
plt.show()
```



4. Discussion

Discuss the results from the previous task: Which network is the most robust against random failures? Explain why this is the case.

- The plot above shows, that with increasing f (increased number of removed nodes), the size of the giant component decreases slowest in the neutral network. That is why the most robust network against random failues is the neutral network. In a neutral network nodes are linked randomly and consequently the density of links is around the average degree.
- In assortative networks hubs tend to link to each other and small-degree nodes tend to connect to small degree nodes.
- In disassortative networks hubs avoid each other. Small-degree nodes tend to connect to hubs, and hubs tend to connect to small-degree nodes (this is called hub-and-spoke character). When removing hubs the network is quickly divided into parts. This explains the rapid reduction of the giant component size in disassortative networks.

Problem 7-4 Random Failures in Uncorrelated Networks

Compute the critical threshold f_c for each of the following degree distributions, under the assumption that the networks do not exhibit any degree correlation.

1. Poisson distribution, i.e.,

$$p_k = e^{-\mu}\frac{\mu^k}{k!}$$

2. Discrete exponential distribution, i.e.,

$$p_k = (1 - e^{-\lambda})e^{-\lambda k}$$

3. Dirac delta distribution, i.e.,

$$p_k = \delta_{k,k_0} = \begin{cases} 1 & \text{if } k = k_0, \\ 0 & \text{otherwise.} \end{cases}$$

Discuss the consequences of your results for network robustness.
Hint: You may use the first and second moment from the lecture or other literature without a proof.

We know f_c can be calculated by:

$$f_c = 1 - \frac{1}{\frac{\langle k^2 \rangle}{\langle k \rangle} - 1}$$

1. For the poisson distribution, we receive:

$$f_c = 1 - \frac{1}{\frac{\mu^2+\mu}{\mu} - 1} = 1 - \frac{1}{\mu}$$

   This means the higher μ the more robust the network is towards random failures. If $\mu \to \infty$ we receive maximum robustness as all nodes would theoretically need to fail for the network to be considered fragmented (even if this does not make sense since there would not be a network present anymore).

2. For the discrete exponential distribution, we receive (using Wolfram Alpha):

$$\langle k \rangle = \frac{(1 - e^{-\lambda})}{\lambda^2}$$

$$\langle k^2 \rangle = \frac{2(1 - e^{-\lambda})}{\lambda^3}$$

$$f_c = 1 - \frac{1}{\frac{2}{\lambda} - 1}$$

   This means the lower λ the more robust the network is towards random failures. If $\lambda \to 0$ we receive maximum robustness.

3. For the dirac delta distribution, we receive:

$$f_c = 1 - \frac{1}{\frac{k_0^2}{k_0} - 1} = 1 - \frac{1}{k_0 - 1}$$

   Similar to the poisson distribution, the dirac delta distribution becomes more robust towards random failures the higher k_0 is. This becomes maximum for clique like structures.