# Bioinformatics III

**Second Assignment**

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

April 27, 2018

## Exercise 2.1: The scale-free network

(a) Listing 1 shows source code.

Listing 1: Example Listing of source code

```
import random
from AbstractNetwork import AbstractNetwork
import Tools
from Node import Node

class ScaleFreeNetwork(AbstractNetwork):
    """Scale-free network implementation of AbstractNetwork"""
    degreeSum = 0

    def __createNetwork__(self, amount_nodes, amount_links):
        """
        Create a network with an amount of n nodes, add m links per iteration step
        for n nodes:
            for m links:
                link node to other nodes
        """
        random.seed()
        numOfNodes = 0
        linksPerIteration = (amount_links-3)/(amount_nodes-3) if amount_nodes > 3 else 1
        #generate n nodes
        while numOfNodes < amount_nodes:
            node = Node(numOfNodes)
            self.appendNode(node)
            numOfNodes += 1
            #make first three nodes fully connected
            if numOfNodes == 2:
                self.__connectNode__(numOfNodes, 1)
            if numOfNodes == 3:
                self.__connectNode__(numOfNodes, 2)
            #link following nodes
            if numOfNodes > 3:
                self.__connectNode__(numOfNodes, linksPerIteration)

    def __connectNode__(self, numOfNodes, linksPerIteration):
        """
        Connect an existing node to m other nodes
        :param numOfNodes: current amount of nodes
        :param linksPerIterations: number of links that should be added in this iteration
        """
        numOfLinks = 0
        node1 = self.getNode(numOfNodes - 1)
        # add n links per iteration
        while numOfLinks < linksPerIteration:
            #choose second node randomly
            nodeid2 = 0
```

```
            r = random.random()
            while nodeid2 < numOfNodes-1:
                node2 = self.getNode(nodeid2)
                #determine probability to choose node
                if self.degreeSum != 0 and node2.degree() != 0:
                    pi = float(node2.degree()) / (self.degreeSum - node1.degree())
                else:
                    pi = float(1)
                if not node1.hasLinkTo(node2):
                    #choose node with probability pi
                    if r < pi:
                        numOfLinks += 1
                        self.degreeSum += 2
                        node1.addLinkTo(node2)
                        node2.addLinkTo(node1)
                        break
                r -= pi
                nodeid2 += 1
```

(b) Listing 2 shows source code.

Listing 2: Example Listing of source code

```
from ScaleFreeNetwork import ScaleFreeNetwork
from RandomNetwork import RandomNetwork
import Tools
import numpy as np


def computeDegreeDistribution(AbstractNetwork):
    """
    Inits DegreeDistribution with a network and calculate its distribution
    """
    # one further entry since 0 is degree 0 is included
    histogram = [0.0] * (AbstractNetwork.maxDegree() + 1)
    # increment degree distribution
    for i in range(0, AbstractNetwork.size()):
        histogram[AbstractNetwork.getNode(i).degree()] += 1.0
    # turn it into a real distribution
    for i in range(0, len(histogram)):
        histogram[i] /= float(AbstractNetwork.size())
    return histogram


def comparison1():
    """
    Compares the degree distribution of a network with 1000 nodes to one with 10000 nodes
    """
    net1 = ScaleFreeNetwork(1000, 1997)
    net2 = ScaleFreeNetwork(10000, 19997)
    hist1 = computeDegreeDistribution(net1)
    hist2 = computeDegreeDistribution(net2)
    histograms = list()
    legend = list()
    histograms.append(hist1)
    legend.append("network with 1000 nodes")
    histograms.append(hist2)
    legend.append("network with 10000 nodes")
    Tools.plotDistributionComparisonLogLog(histograms, legend, "Task 1 b)")


def comparison2():
    """
        Compares the degree distribution of a scale-free network to a random
        network with the same amount of nodes
    """
    net1 = ScaleFreeNetwork(1000, 1997)
    net2 = RandomNetwork(1000, 1997)
```

```
    hist1 = computeDegreeDistribution(net1)
    hist2 = computeDegreeDistribution(net2)
    histograms = list()
    legend = list()
    histograms.append(hist1)
    legend.append("scale-free network")
    histograms.append(hist2)
    legend.append("random network")
    Tools.plotDistributionComparisonLogLog(histograms, legend, "Task 1 b)")


def determineGamma():
    """
    Fits the theoretical distribution of a scale-free network to the degree
    distribution of a scale-free network using the Kolmogorov-Smirnov distance.
    :return: gamma that fits best to the degree distribution of a scale-free network
     with 10 000 nodes and two new links per iteration
    """
    net1 = ScaleFreeNetwork(10000, 19997)
    hist1 = computeDegreeDistribution(net1)
    mindist = float("inf") #minimal distance between theoretical and empirical network
    bestgamma = 0
    #try using gammas between 1 and zero in 0.1 steps
    for gamma in np.arange(1, 3, 0.1):
        #generate thertical distribution with parameter gamma
        hist2 = Tools.getScaleFreeDistributionHistogram(gamma, 10000)
        #compute distance using Kolgomorov-Smirnov distance
        dist = Tools.simpleKSdist(hist1, hist2)
        #set minimal distance
        if dist < mindist:
            mindist = dist
            bestgamma = gamma
    histograms = list()
    histograms.append(hist1)
    histograms.append(hist2)
    legend = list()
    legend.append("empirical distribution")
    legend.append("optimal distribution")
    Tools.plotDistributionComparisonLogLog(histograms, legend, "Task 1 c)")
    print(bestgamma)
```

Both degree distributions, the one for 1000 and 10000 nodes, follow the same distribution. Only the network with more nodes has some nodes with a higher degree than the other network which seems to result from the higher number of nodes. Figure 1 shows the plot.

We also compared a 1000 node scale-free network to a random network of the same size. Unfortunately, because of the higher runtime of the random network generation, we could not finish calculating the corresponding plot.

(c) Listing 3 shows source code.

Listing 3: Example Listing of source code

```
import matplotlib.pyplot as plt
import math

def plotDistributionComparison(histograms, legend, title):
    '''
    Plots a list of histograms with matching list of descriptions as the legend
    '''
    # determine max. length
    max_length = max(len(x) for x in histograms)

    # extend "shorter" distributions
    for x in histograms:
        x.extend([0.0]* (max_length-len(x)) )

    # plots histograms
```

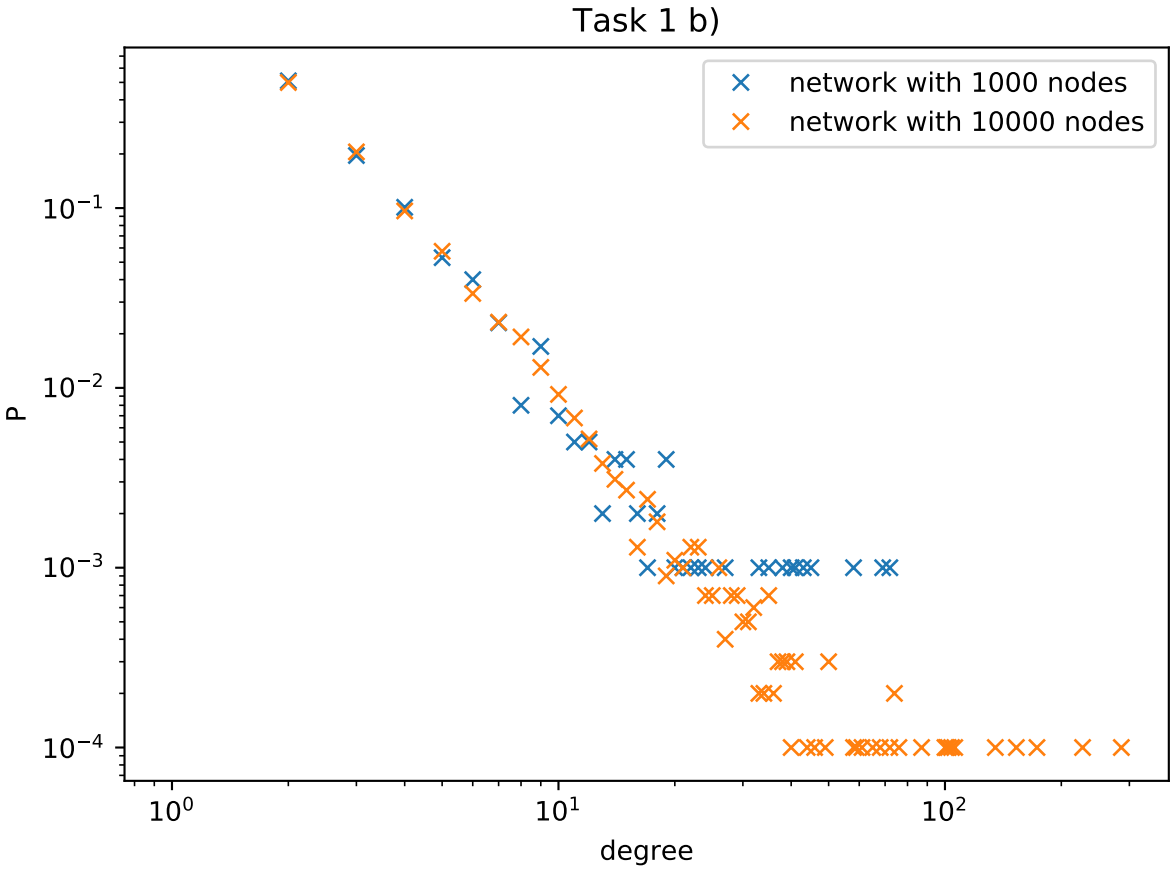


Figure 1: Comparison of two scale-free networks

```
    for h in histograms:
        plt.plot(range(len(h)), h, marker = 'x')

    # remember: never forget labels!
    plt.xlabel('degree')
    plt.ylabel('P')

    # you don't have to do something stuff here
    plt.legend(legend)
    plt.title(title)
    plt.tight_layout()
    plt.show()

def plotDistributionComparisonLogLog(histograms, legend, title):
    '''
    Plots a list of histograms with matching list of descriptions as the legend
    '''
    ax = plt.subplot()
    # determine max. length
    max_length = max(len(x) for x in histograms)

    # extend "shorter" distributions
    for x in histograms:
        x.extend([0.0]* (max_length-len(x)) )

    ax.set_xscale("log")
    ax.set_yscale("log")

    # plots histograms
    for h in histograms:
        ax.plot(range(len(h)), h, marker = 'x', linestyle='')

    # remember: never forget labels!
    plt.xlabel('degree')
    plt.ylabel('P')

    # you don't have to do something stuff here
    plt.legend(legend)
    plt.title(title)
    plt.tight_layout()
    plt.show()

def getScaleFreeDistributionHistogram(gamma, k):
    '''
    Generates a Power law distribution histogram with slope gamma up to degree k
    '''
    histogram = list()
    histogram.append(0)
    for i in range(1, k):
        histogram.append(1.0 / math.pow(i, gamma))
    return histogram


def simpleKSdist(histogram_a, histogram_b):
    '''
    Simple Kolmogorov-Smirnov distance implementation
    '''
    dist = list()
    F1 = {0: histogram_a[0]}
    F2 = {0: histogram_b[0]}
    for x in range(1, len(histogram_a)):
        F1[x] = F1[x-1] + histogram_a[x]
        F2[x] = F2[x - 1] + histogram_b[x]
        dist.append(abs(F1[x] - F2[x]))
    return max(x for x in dist)
```

Comparing the empirical and the theoretical distributions, one may see the first third of the

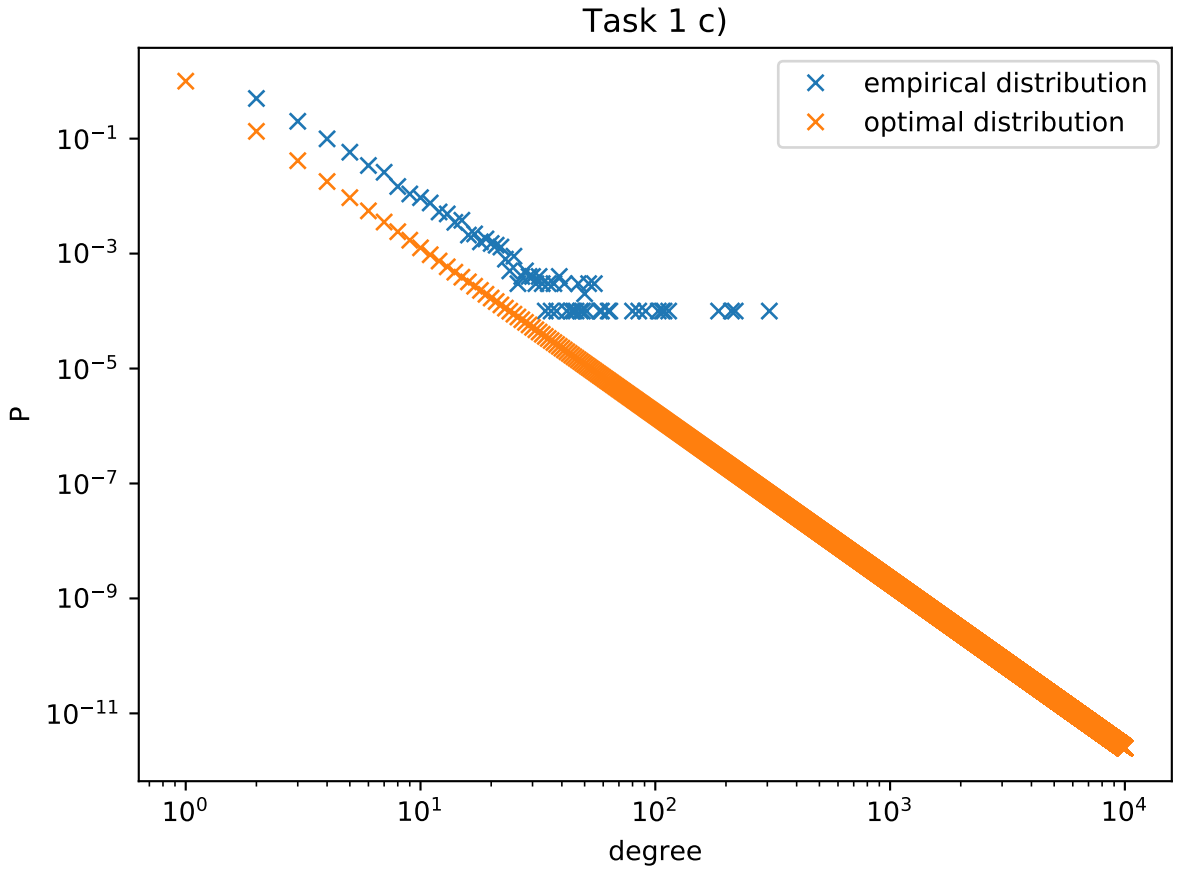


Figure 2: Comparison of empirical to theoretical network

graph fit well, whereas the rest of the empirical distribution is very differently distributed. We determined a gamma value of 1.8. The quality of our fit isn't very high. Maybe it could be improved by computing a average distance between each value. Figure 2 shows the plot.

## Exercise 2.2: Real-world network

(a) File sharing services like Google Drive form clustered networks, which are clustered by the users which have access to a file. Every time a user adds a new file, a directed link connects a new file to an user.

(b) Social networks like Facebook, Twitter and so on may be represented as undirected scale-free networks because people with many friends are more likely to get new friends because they know many people. Moreover a connection between two users is not directed because both users have to accept a friend request.
A social network can also be represented as a clustered network, whereby the clustered are made of different groups of friends.

(c) Broadcasting networks may form hierarchical or clustered networks. In the case of a hierarchical network, we assume that one broadcaster sends data to multiple other services which publish the data. The network could be clustered by the receiver, which receive data from the same broadcaster. A directed node connects each broadcaster to its receiver.

## Exercise 2.3: Real interaction networks

(a)

(b)

(c)

(d)