Bioinformatics III

Second Assignment

Max Jakob (2549155)
Carolin Mayer (2552320)

April 27, 2018

Exercise 2.1: The scale-free network

(a) For the implementation of the `ScaleFreeNetwork` class, the given classes `AbstractNetwork`, `Node` and `Tools` were used. They are listed below. Only in `Tools`, some changes have been made, including a plot function, a poisson distribution function and the two requested functions `getScaleFreeDistributionHistogram(gamma, k)` and `simpleKSdist(histogram-a, histogram-b)`.

Listing 1: Source code of Node.py

```
# Node class, assignment 1
class Node:
    def __init__(self, identifier):
        """
        Sets node id and initialize empty node list that references its connected nodes
        """
        self.id = identifier
        self.nodelist = []

    def hasLinkTo(self, node):
        """
        Returns True if this node is connected to node asked for,
        False otherwise
        """
        return (node in self.nodelist)

    def addLinkTo(self, node):
        """
        Adds link from this node to parameter ode (only if there is no link connection already)
        does not automatically care for a link from parameter node to this node
        """
        if (~self.hasLinkTo(node)):
            self.nodelist.append(node)

    def degree(self):
        """
        Returns degree of this node
        """
        return len(self.nodelist)

    def __str__(self):
        """
        Returns id of node as string
        """
        return str(self.id)

    def getNodeSet(self):
        return set(self.nodelist)

    def removeNode(self,node):
```

```
40          self.nodelist.remove(node)
```

Listing 2: Source code of AbstractNetwork.py

```
0  from Node import Node

   class AbstractNetwork:
       """Abstract network definition, can not be instantiated"""

5      def __init__(self, amount_nodes, amount_links):
           """
           Creates empty nodelist and call createNetwork of the extending class
           """
           self.nodes = {}
10         self.__createNetwork__(amount_nodes, amount_links)

       def __createNetwork__(self, amount_nodes, amount_links):
           """
           Method overwritten by subclasses, nothing to do here
15         """
           raise NotImplementedError

       def appendNode(self, node):
           """
20         Appends node to network
           """
           self.nodes[node.id] = node

       def maxDegree(self):
25         """
           Returns the maximum degree in this network
           """
           return max([x.degree() for x in self.nodes.values()])

30     def size(self):
           """
           Returns network size
           """
           return len(self.nodes)
35
       def __str__(self):
           return "This network has %6d nodes." % (len(self.nodes))


40     def getNode(self, identifier):
           """
           Returns node according to key
           """
           if identifier not in self.nodes:
45             self.nodes[identifier] = Node(identifier)

           return self.nodes[identifier]
```

Listing 3: Source code of Tools.py

```
0  import matplotlib.pyplot as plt
   import math

   import numpy as np


5
   def plotDistributionComparison(histograms, legend, title):
       '''
       Plots a list of histograms with matching list of descriptions as the legend
       '''
10     # 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
    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):

    '''
        function to create Scale Free distribution based on the power law distribution, normal
    '''
    histogram = [0.0] * k
    for i in range(1,k):
        histogram[i] = math.pow(i, -gamma)
    num = np.sum(histogram)
    return [i / num for i in histogram]


def simpleKSdist(histogram_a, histogram_b):
    '''
        function to compute the Kolmogorov-Smirnov distance
    '''

    a_sum = np.cumsum(histogram_a)
    b_sum = np.cumsum(histogram_b)
```

```
80      index_max_deviate = -1
        for i in range(2, histogram_a.__len__()):   # start at degree 2, because no node can have d
            deviate = abs(a_sum[i] - b_sum[i])
            if deviate > index_max_deviate:
                index_max_deviate = deviate
85
        return index_max_deviate


    '''
90      Function to plot a the degree distribution of the human interaction network
        takes hisogram
        creates two plots, one with a shrunken x-axis, one with all entries
    '''
    def plotHumanNetwork(hist):
95      axes = plt.gca()
        axes.set_xlim([0, 100])
        plt.plot(hist, marker='x')

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

        # you don't have to do something stuff here
        plt.legend(["Human"])
105     plt.title("Plot_1")
        plt.tight_layout()
        plt.savefig("Plot2_1")

        axes = plt.gca()
110     axes.set_xlim([0, 2369])
        plt.plot(hist, marker='x')

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

        # you don't have to do something stuff here
        plt.legend(["Human"])
        plt.title("Plot_2")
120     plt.tight_layout()
        plt.savefig("Plot2_2")


    def getPoissonDistributionHistogram(num_nodes, num_links, k):
125     '''
        Generates a Poisson distribution histogram up to k
        '''
        lam = 2 * num_links / num_nodes
        res = [0]*k
130     res[0] = math.exp(-lam)
        for i in range(1, k):
            res[i] = lam/k * res[i-1]
        return res
```

The implementation of the `ScaleFreeNetwork` is shown below. To gain a high performance of the network build up, we created a list of all nodes. This list contains each node as many times as links it contains, thats to say the degree of the node. To sample now with wight respective to the degree, one only has to randomly draw a node out of this list, as it represents the overall degree distribution. The list is updated every time a link is added by the two link-corresponding nodes. A they only have to be added at the end, this implementation results in high performance.

Listing 4: Source code of ScaleFreeNetwork.py

```python
import random
from AbstractNetwork import AbstractNetwork
from Node import Node
import numpy as np

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

    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()

        for i in range(0, 3):
            AbstractNetwork.appendNode(self, node=Node(i))

        n0 = AbstractNetwork.getNode(self, 0)
        n1 = AbstractNetwork.getNode(self, 1)
        n2 = AbstractNetwork.getNode(self, 2)
        n0.addLinkTo(n1)
        n1.addLinkTo(n2)
        n2.addLinkTo(n0)
        n0.addLinkTo(n2)
        n1.addLinkTo(n0)
        n2.addLinkTo(n1)

        linked_nodes = [0,0,1,1,2,2]
        limit = amount_links

        for i in range(3, amount_nodes):
            network_size = self.size()
            if amount_links > network_size:
                limit = network_size

            new_node = AbstractNetwork.getNode(self, i)
            for j in range(0, limit):
                ns = random.randint(0, linked_nodes.__len__()-1)
                while new_node.hasLinkTo(linked_nodes[ns]):
                    ns = random.randint(0, linked_nodes.__len__()-1)
                n = AbstractNetwork.getNode(self, linked_nodes[ns])
                new_node.addLinkTo(n)
                n.addLinkTo(new_node)
                linked_nodes.append(i)
                linked_nodes.append(linked_nodes[ns])

    def getDegreeDist(self):
        size = self.maxDegree() + 1
        hist = [0] * size
        for node in self.nodes:
            i = self.nodes[node].nodelist.__len__()
            hist[i] = hist[i] + 1
        num = np.sum(hist)
        return [i / num for i in hist]

    def getNumLinks(self):
        size = self.maxDegree() + 1
        hist = [0] * size
        for node in self.nodes:
            i = self.nodes[node].nodelist.__len__()
            hist[i] = hist[i] + 1
        return np.sum(hist)
```

(b) The degree distribution of two scale free networks are shown in Figure 1. The main difference

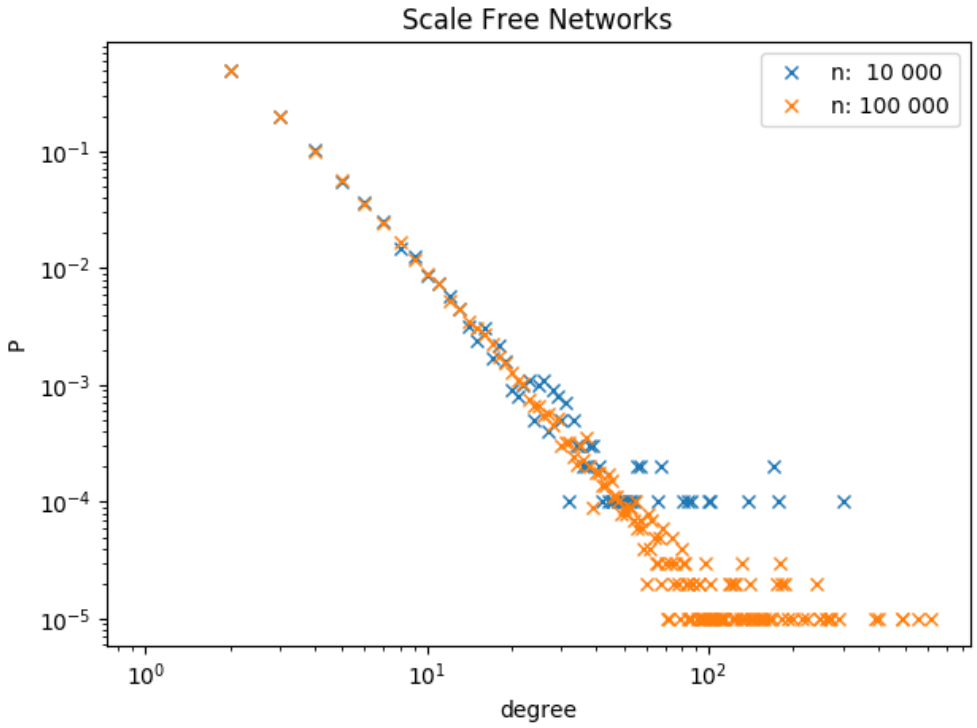

Figure 1: Showing the degree distribution for two random networks in a double logarithmic scale.

between the two curves in the plot is the respective 'depth'. A scale free network with 100 000 nodes contains a higher maximal degree, which 'stretches' the distribution. Apart from that, the two distributions are the same.
Figure 2 shows the scale free degree distribution vs. the degree distribution of a random network. We can observe, that the probability of a certain degree falls by the size of the degree. This probability falls much faster in the random network. In the scale free network, it also falls, but this is not clearly observable in this plot, as the random network distribution falls that quick in respect.

(c) Figure 3 shows the comparison of the observed scale free network degree distribution vs the theoretical power law distribution. The best value for γ between 1.0 and 3.0 was used by iterating over this interval with step size 0.1. The best result was obtained by taking the minimal KS distance of every theoretical distribution to the observed one. The two shown distributions look similar, but rotated a little. The slope may be different because we start our scale free network with only 3 nodes. If we start with more initial nodes, we would also obtain a different degree distribution and thus a different slope in the curve. But the general shape of our scale free distribution looks quite similar to the the power law distribution. Another reason for our mismatch may be the fact that we are missing the normalization constant C in our power law computation.

Exercise 2.2: Classify real-world network examples

(a) File sharing services as Dropbox and Google drive follow a clustered degree distribution. As one can upload files and control the access rights, one usually shares documents within a special group. The persons inside the group also share their content with the group, which makes the overall network a clustered one.

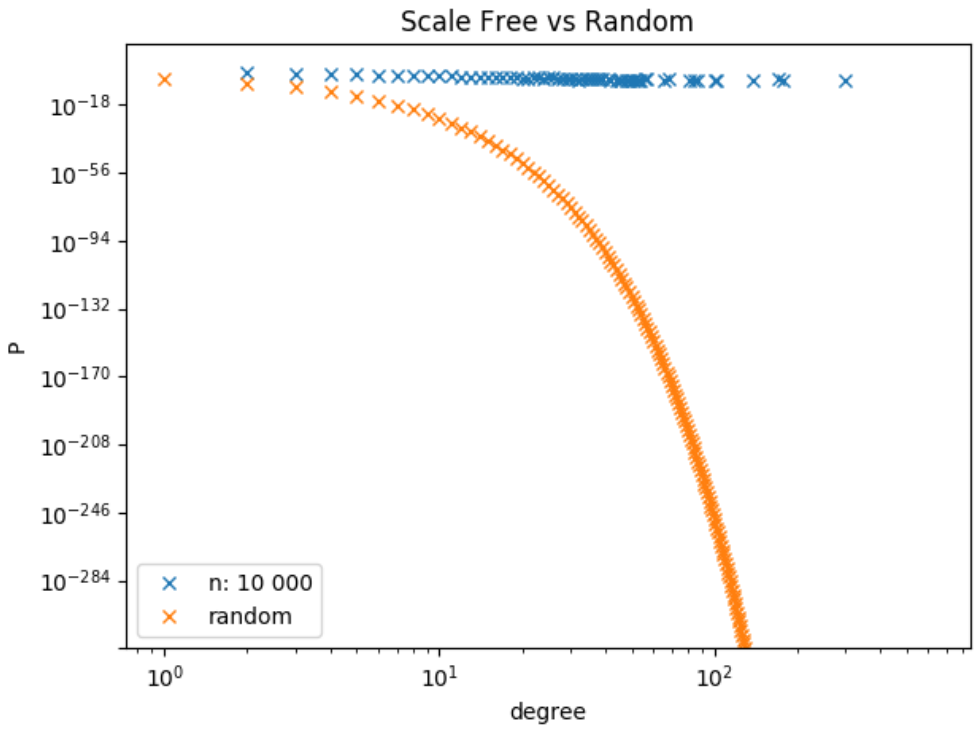

Figure 2: Comparison between the degree distribution of a scale free network and a random network in double logarithmic scale.

(b) Social networks usually follow a scale free distribution. One usually follows friends or famous people in social networks. As a famous person got a lot of links (friends/followers), their 'degree' increases by the number of persons that join the network. This makes it scale free looking like network.

(c) Satellite or cable television distribution follows a hierarchical structure. The content gets created a special point. To reach all costumers, the content is send to different server locations all over the world. These send the content to local servers which send it to every consumer. This forwarding results in a hierarchical structure, as ever costumer only got one access point which got several other access points which all connect to the upper global server.

Exercise 2.3: Real interaction networks

(a) The implementation of the BioGrid reader is listed below. To store the data, a class structure of organisms and genes was used, which can also be found in `BioGRIDReader`.

Listing 5: Source code of BioGRIDReader.py

```
import re
from operator import itemgetter


class BioGRIDReader:
    '''Reads BioGRID tab files'''

    def __init__(self, filename):
        '''
        Initialization, read in file and build any data structure that makes you happy
```

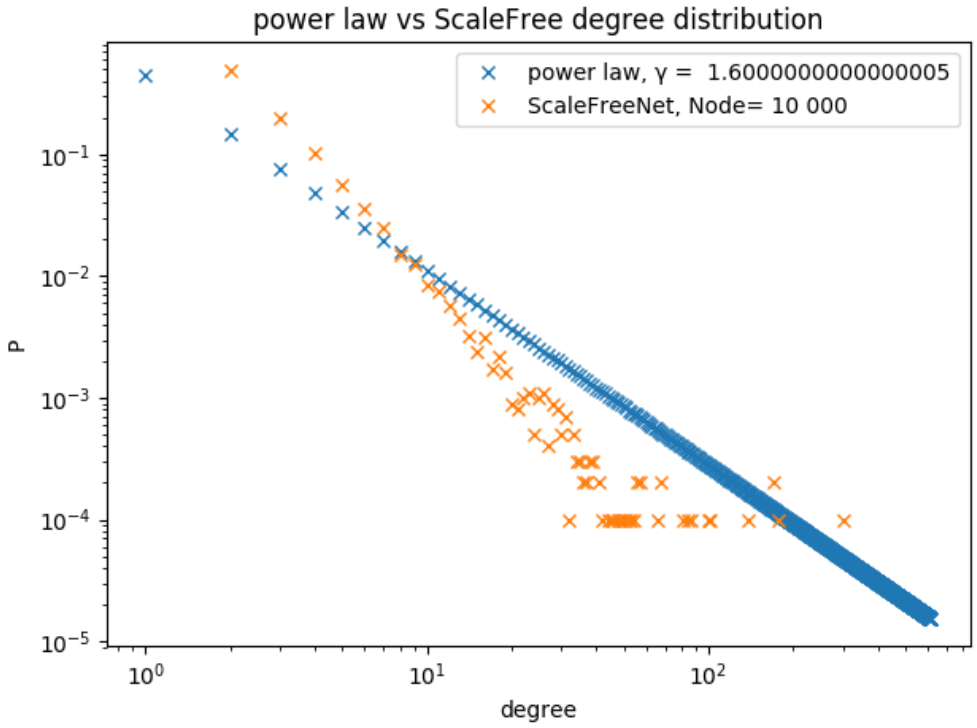

Figure 3: Comparison between degree distribution for a scale free network and a power law distribution in double logarithmic scale.

```
        '''
        self.organisms = {}  # dictionary of all organisms in the data
        self.file = filename
        file = open(filename, "r")  # open file to read
        read = 1
        for line in file:
            if line.startswith("INTERACTOR_A"):   # skip intro in file until header of table o
                read = 0
                continue
            if read == 1:
                continue

            temp = re.split(r'\t+', line)  # split every line
            if (temp.__len__() != 11):
                continue
            geneA = temp[2]      # extract only name of genes and both organisms
            geneB = temp[3]
            orgA = int(temp[9])
            orgB = int(temp[10])
            if (orgA == orgB):      # add to the according dictionary if the two organisms are t
                if (self.organisms.__contains__(orgA)):
                    self.organisms[orgA].addInteraction(geneA, geneB)
                else:
                    t = Organism(orgA)
                    self.organisms[orgA] = t

    '''
        function to get the most 'n' abundant Organisms from the data based on simple linkage c
    '''
    def getMostAbundantTaxonIDs(self, n):
        all = []
        for key in self.organisms:      # export all organisms with count
```

```
            all.append([key, self.organisms[key].number_of_interactions()])
        all.sort(key=itemgetter(1), reverse=True)   # sort by counting

        # print n top organisms
        print("Top " + n.__str__() + " Organism by number of gene interactions")
        for i in range(0, n):
            print((i + 1).__str__() + ". Organism: " + all[i][0].__str__() + ";\tCount: " + all

        return all[0:n]

    '''
        Function to print the 'n' Genes with highest degree for a given taxon ID
    '''
    def InteractionNetwork_by_taxID(self, n, id):
        all = []
        if not (self.organisms.__contains__(id)):   # print if id not in set
            print("Taxon_ID '" + id.__str__() + "' not found in data set...")
            return

        org = self.organisms[id]
        for gene in org.interactions:   # export every gene with degree
            all.append([gene, org.interactions[gene].links.__len__()])
        all.sort(key=itemgetter(1), reverse=True)   # sort by degree

        # printing top n genes
        print("Top " + n.__str__() + " Genes by degree from Organism " + id.__str__())
        for i in range(0, n):
            print((i + 1).__str__() + ". Gene: " + all[i][0].__str__() + ";\tDegeree: " + all[i

    '''
        Function to write a file with all gene interactions in the read dataset
        according to the given taxon id
        Format: (tab delimited)
        Organism_Id     Gene_ID     List of Gene_ID where a interaction was found
    '''
    def writeInteractionFile(self, taxon_id, filename):
        file = open(filename, "w")
        if not (self.organisms.__contains__(taxon_id)):  # print if id not in set
            print("Taxon_ID '" + taxon_id.__str__() + "' not found in data set...")
            return
        org = self.organisms[taxon_id]
        for gene in org.interactions:
            g = org.interactions[gene]
            links = ""
            first = 0
            for i in g.links:
                if (first == 0):
                    links = links + i
                    first = 1
                else:
                    links = links + "\t"
                    links = links + i

            file.write(org.ncbi.__str__() + "\t" + g.name.__str__() + "\t" + links + "\n")
        file.close()

'''
    Class to represent an organism
'''
class Organism:

    def __init__(self, ncbi_id):
        self.ncbi = ncbi_id     # id
        self.interactions = {}  # list if genes with interactions
        self.num = 0            # number of interactions

    '''
```

```
        add interaction between two given genes
    '''
    def addInteraction(self, geneA, geneB):
        self.num += 1   # increase counter of interactions
        if not self.interactions.__contains__(geneA):   # if genes not in dictionary yet, creat
            t = Gene(geneA)
            self.interactions[geneA] = t
        if not self.interactions.__contains__(geneB):   # if genes not in dictionary yet, creat
            t = Gene(geneB)
            self.interactions[geneB] = t
        self.interactions[geneA].addLink(geneB)     # add link
        self.interactions[geneB].addLink(geneA)     # add link

    '''
        function to get the number of interactions
    '''
    def number_of_interactions(self):
        return self.num


'''
    class to represent a gene
'''
class Gene:

    '''
        got name and a list of genes, which it interacts with
    '''
    def __init__(self, name):
        self.name = name
        self.links = []

    '''
        add link to gene, if link does not exist yet
    '''
    def addLink(self, name):
        if(name == self.name):  # no self links are allowd
            return
        if not (self.links.__contains__(name)):
            self.links.append(name)
```

(b) For implementation look at Listing 5. Top 5 Organism by number of gene interactions are listed here:

	Organism	Taxon ID	count
1.	yeast	559292	702148
2.	human	9606	386191
3.	E.coli	316407	184019
4.	yeast	284812	71990
5.	Drosophila melanogaster	7227	67728

This order is not quit surprising, as all accept of human are really simple organism, where research and analysis can be applied easily. Human is listed so high, as it is a big organism with a lot of genes and thus also gene interactions. Also it is the main field of interest in research to understand human gene interactions, as they are the most interesting in respect to drug development and fundamental understanding of the human being.

(c) The human interaction network consists of 22 463 genes and 386 191 known interactions. The top active genes in the sens of mostly connected in the network are listed below.

	Gene	Degree
1.	TRIM25	2367
2.	APP	2097
3.	NTRK1	1942
4.	ELAVL1	1777
5.	XPO1	1214
6.	CUL3	1207
7.	EGFR	1193
8.	NXF1	1122
9.	MOV10	1010
10.	TP53	1009

We obtained a short description for EGFR (Epidermal Growth Factor Receptor) from its gene cards entry:
The protein encoded by this gene is a transmembrane glycoprotein that is a member of the protein kinase superfamily. This protein is a receptor for members of the epidermal growth factor family. EGFR is a cell surface protein that binds to epidermal growth factor. Binding of the protein to a ligand induces receptor dimerization and tyrosine autophosphorylation and leads to cell proliferation. Mutations in this gene are associated with lung cancer.
It is not surprising to that this gene have a lot of known gene interactions, as it is a main part in the growth of the human epidermal cells. These cells are spread all over the human body and therefor this gene have to interact with several different other genes.

(d) The implementation of the `GenericNetwork` is listed below. For the implementation of the `writeInteractionFile(taxon-id, filename)` please consider Listing 5. To run our script and obtain all shown plots, please run Listing 7.

Listing 6: Source code of GenericNetwork.py

```
from AbstractNetwork import AbstractNetwork
import re
import numpy

'''
    class to extend AbstractNetwork to read network from file
'''
class GenericNetwork(AbstractNetwork):

    def __init__(self, filename):
        self.filename = filename    # path to file
        self.nodes = {}             # node dict
        self.__createNetwork__()    # build network

    '''
        function to read file and build network
    '''
    def __createNetwork__(self):
        file = open(self.filename, "r")
        for line in file:
            temp = re.split(r'\t+', line)   # split line after tab
            if (temp.__len__() < 2):        # if less than 2 elements, ignore
                continue
            geneA = temp[1]                 # gene, part of all following interactions
            n1 = AbstractNetwork.getNode(self, geneA)   # node from network, according to the g
            for i in range(2, temp.__len__()):          # iterate over all genes in list, add
                # can use getNode(), as it adds a new node automatically if node is not in netw
                ntemp = AbstractNetwork.getNode(self, temp[i])
                if(n1.id == ntemp.id):                          # not self links allowed
                    continue
                if not (n1.hasLinkTo(ntemp)):                   # add link
                    n1.addLinkTo(ntemp)
                if not (ntemp.hasLinkTo(n1)):                   # add link
```

```
                    ntemp.addLinkTo(n1)

        '''
            function to obtain the normalized degree distribution of the network
        '''
        def getDegreeDist(self):
            size = self.maxDegree() + 1
            hist = [0] * size
            for node in self.nodes:
                i = self.nodes[node].nodelist.__len__()
                hist[i] = hist[i] + 1
            num = numpy.sum(hist)
            return [i / num for i in hist]
```

Listing 7: Source code of Exercise2.3.py

```
from GenericNetwork import GenericNetwork
from BioGRIDReader import BioGRIDReader
import Tools


'''
    Main:
        call to run exercise 2.3 and create all content for it
'''
if __name__ == '__main__':

    # file spec, put absolute or relative path here
    filename = "BIOGRID-ALL-3.4.159.tab.txt"
    print("\n\tstart reading data from '" + filename + "' ...")
    # reads the BioGrid using the BIOGRIDReader
    grid = BioGRIDReader(filename)
    # get 5 most abundant organisms
    grid.getMostAbundantTaxonIDs(5)
    # prints the 10 most interacting genes in human(9606)
    grid.InteractionNetwork_by_taxID(n=10, id=9606);
    # writes file of all gene interactions in humans to out.txt
    grid.writeInteractionFile(9606, "out.txt")

    # creates real interaction network for humans parsing out.txt
    net = GenericNetwork("out.txt")
    # print net to see it works
    print(net)
    # get degree distribution
    hist = net.getDegreeDist()
    # plot degree distribution in two plots, a shrunken and a full version
    Tools.plotHumanNetwork(hist)
```

In Figure 4 two plots are shown. These represent the degree distribution of the human gene interaction network from `BioGRID`. The two plots only differ in scaling. The network follows more a ScaleFree distribution as to a Random distribution network. A random network would have a more bell shaped distribution in the area of degrees from 0 to 100. A ScaleFree distribution looks like an L shape (strong peak at the beginning, than flattens out fast), which is clearly observable at this plot of the human gene interaction network.

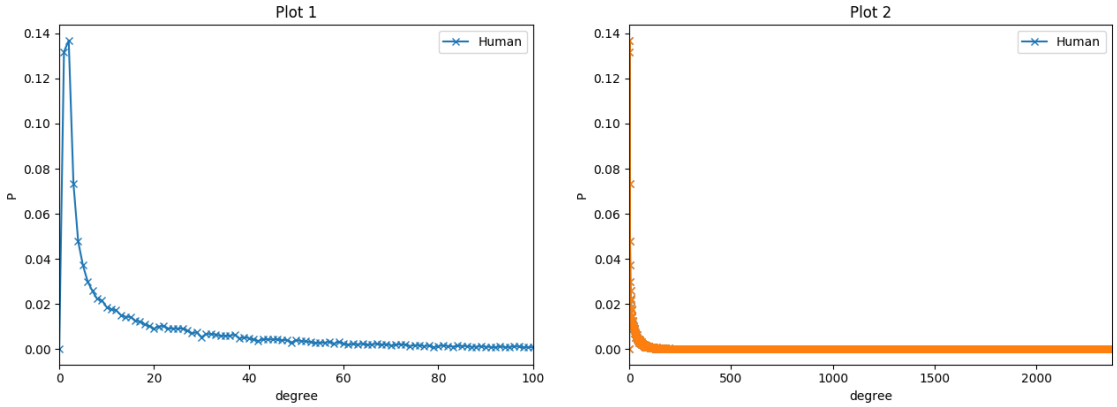

Figure 4: The degree distribution of the human gene interaction network, left for degree 0-100, right all degrees shown (22463 in total).