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
```

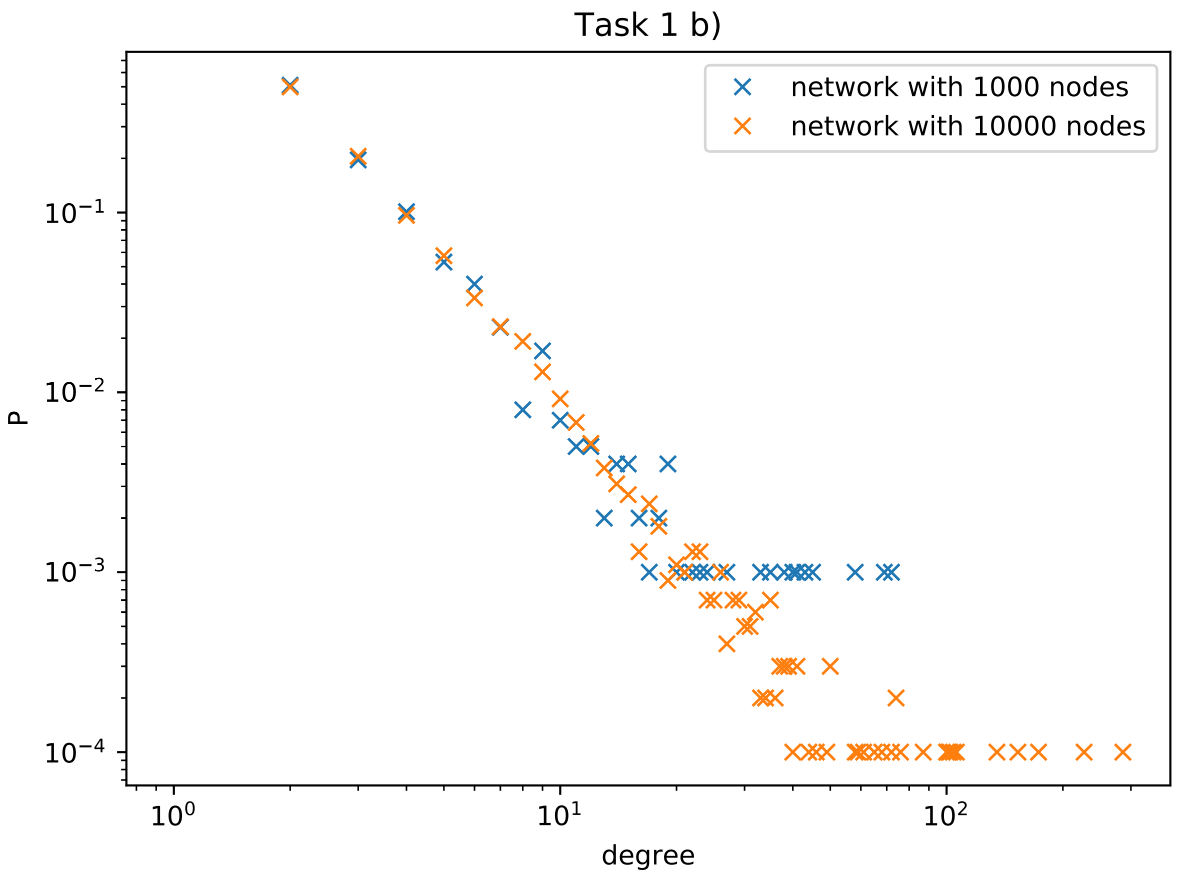

Figure 1: Comparison of two scale-free networks

```
    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)

def poisson(k, l):
    '''
    Compute the poisson entry for k and lambda (l)
    '''
    k = float(k)
    l = float(l)
    if (k == 0):
        return(math.exp(-1.0*l))
    else:
        return (l/k)*poisson(k-1.0,l)

def getPoissonDistributionHistogram(num_nodes, num_links, k):
    '''
    Generates a Poisson distribution histogram up to k
    '''
    poissonHist = []
    lambda_ = 2.0*(float(num_links))/float(num_nodes)
    print "Lambda:", lambda_
    for i in range(0,num_links):
        if(i <= k):
                poissonHist.append(poisson(i, lambda_))
    return poissonHist
```

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

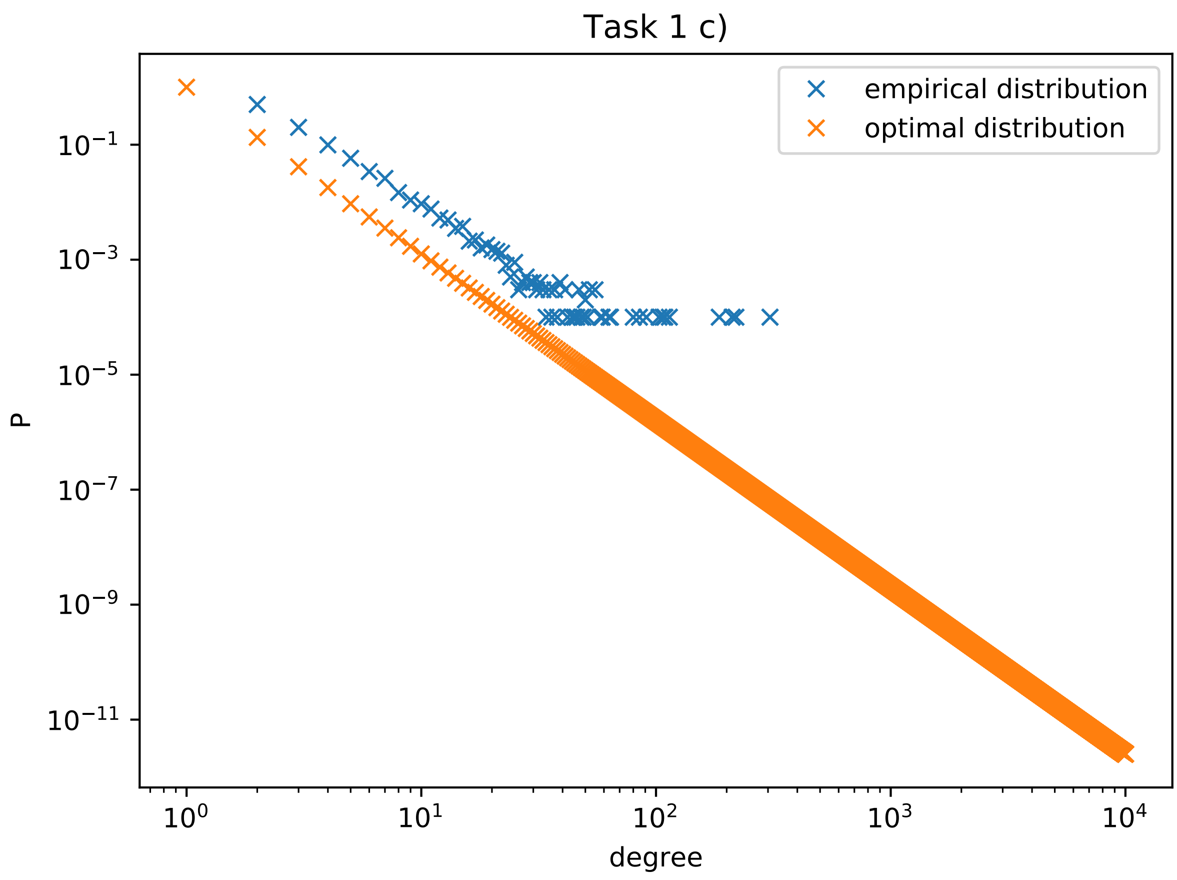

Figure 2: Comparison of empirical to theoretical network

Exercise 2.3: Real interaction networks

(a) Listing 4 shows source code of BioGRIDReader. To retrive all interactions of an organism, use function getInteractions(taxonID)

Listing 4: Example Listing of source code

```python
from os.path import exists
from sets import Set

class BioGRIDReader:
    '''Reads BioGRID tab files'''
    def __init__(self, filename):
        '''
        Initialization, read in file and build any data structure that makes you happy
        '''
        # Use dic of Sets of Tuples
        self.storage = dict()
        if exists(filename):
            # "with" closes the file again after reading
            with open(filename) as openfile:
                # catch table header line
                first = True
                for line in openfile:
                    # get entries of a line as list
                    content = self.separate(line[0:(len(line)-1)])
                    # and store them
                    if first:
                        first = False
                    else:
                        self.insertInteraction(content)
        else:
            print filename, "does not exist"

    def separate(self, line):
        '''
        Split line into list of entries
        '''
        if "\t" in line:
            content = line.split("\t")
            return content
        # return empty list if not splitable
        return []

    def insertInteraction(self, content):
        '''
        Store content of a line in the dict structure
        '''
        # check validity of input
        if len(content) == 11:
            # check if both genes are from the same organism
            if content[9] == content[10]:
                # extract important information
                organism = content[9]
                geneA = content[0]
                geneB = content[1]
                # store the results
                if self.storage.has_key(organism):
                    self.storage[organism].add((geneA, geneB))
                    # avoid reverse duplicated for non-self-loops
                    if geneA != geneB:
                        self.storage[organism].discard((geneB, geneA))
                else:
                    # use a set, it avoids duplicates
                    self.storage[organism] = Set()
                    self.storage[organism].add((geneA, geneB))

    def getInteractions(self, taxonID):
        '''
        Return the number of links between genes of a prganism
        '''
        return self.storage[taxonID]

    def getMostAbundantTaxonIDs(self, n):
        '''
        Returns a list of n (taxonID/number of links) lists with
        the most reported genes interactions per organisms
        '''
        # allow bad paraeters to be handled
        if n > len(self.storage):
            n = len(self.storage)
        if n < 0:
            n = 0
        # use lefmade sort algorithm
        # iterate over organisms and store the number of interaction
        # use a decreasing order
        counter = []
        first = True
        for i in self.storage.keys():
            links = len(self.storage[i])
            inserted = False
            # first element must be inserted
            if first:
                counter.append([i, links])
                first = False
            else:
                # find position for insertion
                for j in range(0,len(counter)):
                    if links > counter[j][1]:
                        counter.insert(j,[i, links])
                        inserted = True
                        break
                if not inserted:
                    counter.append([i, links])
        # return the required n organisms with the most interactions
        return counter[0:n]

    def writeInteractionFile(self, taxon_id, filename):
        if taxon_id in self.storage.keys():
            output = open(filename, "w")
            for (a,b) in self.storage[taxon_id]:
                output.write(a + "\t" + b + "\n")
            output.close()
```

(b) Results for organisms with most annotated interactions:

Table 1: List of top five organisms with the most annotated interactions mentioned in the BioGRID

Taxon ID	Organism	Number of annotated interaction
559292	*Saccharomyces cerevisiae S288C*	513254
9606	*Homo sapiens*	275472
316407	*Escherichia coli str. K-12 substr. W3110*	181620
284812	*Schizosaccharomyces pombe 972h-*	58563
7227	*Drosophila melanogaster*	55093

"The genome of Saccharomyces cerevisiae is by far the best studied fungal genome" (https://genome.jgi.doe.gov/Sacce1/Sacce1.home.html. The human genome is known since the start of the 21th century, and often analyzed since that time. *E. coli* cells are one of the most popular model organisms for laboratory experiments. The same holds true for the fruit fly *Drosophila*. Further, *Schizosaccharomyces pombe 972h-*, or fission yeast is a well known model organism too.
So, beside humans, all top 5 listed organisms are popular laboratory model organisms. Humans are not neccessarily model organisms, but clinical research uncovered many intracellular interaction. Therefore, the list is not surprising.

(c) The human interaction network has 275472 annotated interactions and 17087 nodes. The top 10 interacting proteins are:

Table 2: Top 10 interacting proteins of humans, annotated in BioGRID, last colum shows evaluation of the Biogrid webpage (https://thebiogrid.org/)

ID	Protein	Interactions	reported from BioGrid: Unique Interactor (Interactions)
ETG7706	TRIM25	2369	2371(2551)
ETG351	APP	2099	2115(2475)
ETG4914	NTRK1	1944	1943(1999)
ETG1994	ELAVL1	1779	1780(1840)
ETG7514	XPO1	1214	1236(1315)
ETG8452	CUL3	1209	1219(1640)
ETG1956	EGFR	1195	1209(2128)
ETG10482	NXF1	1124	1132(1202)
ETG7157	TP53	1011	1053(3075)
RP11-426L16.2	MOV10	1010	1024(1050)

Hint: The difference in calculated and reported interaction (from BioGRID) can be explained by the fact that our network does not allow multiple links between the same proteins. Further the differences of unique interaction partners can be partly explained by selfloop wich are ignored in the network.
EGFR: This protein is involved in singnaling pathways, which normaly includes many protein interactions. Example here is the MAPK cascade in which the protein is involved. Further, EGFR is involved in the transcription of the Polymerase II. Overall, https://thebiogrid.org/ assignes 70 GO annotations to EGFR, 42 for biological processes, 15 for its function and 13 GO components. All these listed annotations hint on activities in which many proteins are involved. This can explain the many interactions, reported in our network.

(d) The function writeInteractionFile is implemented in the class BioGRIDReader, listings 4. A possible output for human organisms is given in the file human.txt. Listing 5 shows source code of the generic Network.

Listing 5: GenericNetwork class which reads the network from a special file

```
from os.path import exists
from AbstractNetwork import AbstractNetwork
from BioGRIDReader import BioGRIDReader
from AbstractNetwork import AbstractNetwork
from Node import Node

class GenericNetwork(AbstractNetwork):
    def __init__(self, filename):
        """
        Create a network from a file
        """
        self.nodes = {}
        if exists(filename):
            # "with" closes the file again after reading
            with open(filename) as openfile:
                for line in openfile:
                    # get entries of a line as list
                    content = line[0:(len(line)-1)].split("\t")
                    # and store them
                    if len(content) == 2:
                        n1 = self.getNode(content[0])
                        n2 = self.getNode(content[1])
                        n1.addLinkTo(n2)
                        n2.addLinkTo(n1)
        else:
            print filename, "does not exist"


    def printDegreeHigherThan(self, n):
        '''
        Print proteins, with more than n interactions
        '''
        for i in self.nodes:
            if(self.getNode(i).degree() > n):
                print i, self.getNode(i).degree()

    def degreeDistribution(self):
        vector = [0.0]*(self.size()+1)
        for name in self.nodes:
            node = self.getNode(name)
            vector[node.degree()] += 1.0
        for i in range(0,len(vector)):
            vector[i] /= float(self.size())
        return vector
```

The generic network has its own distribution calculation due to an error in the DegreeDistribution class. The degree distribution of our human interaction network can be seen in figure 3. Green colored is the poisson distribution, which we can expect when analyzing a rendom network. It is obvious, that the Human interaction network has a different distribution. The red line, the estimated distribution of a scale-free network on the other hand lokks similar to our distribution. Even if the applied parameters for the red line were not similar to our human distribution.
Therefore, the human interaction network behaves more like a scale-free network.

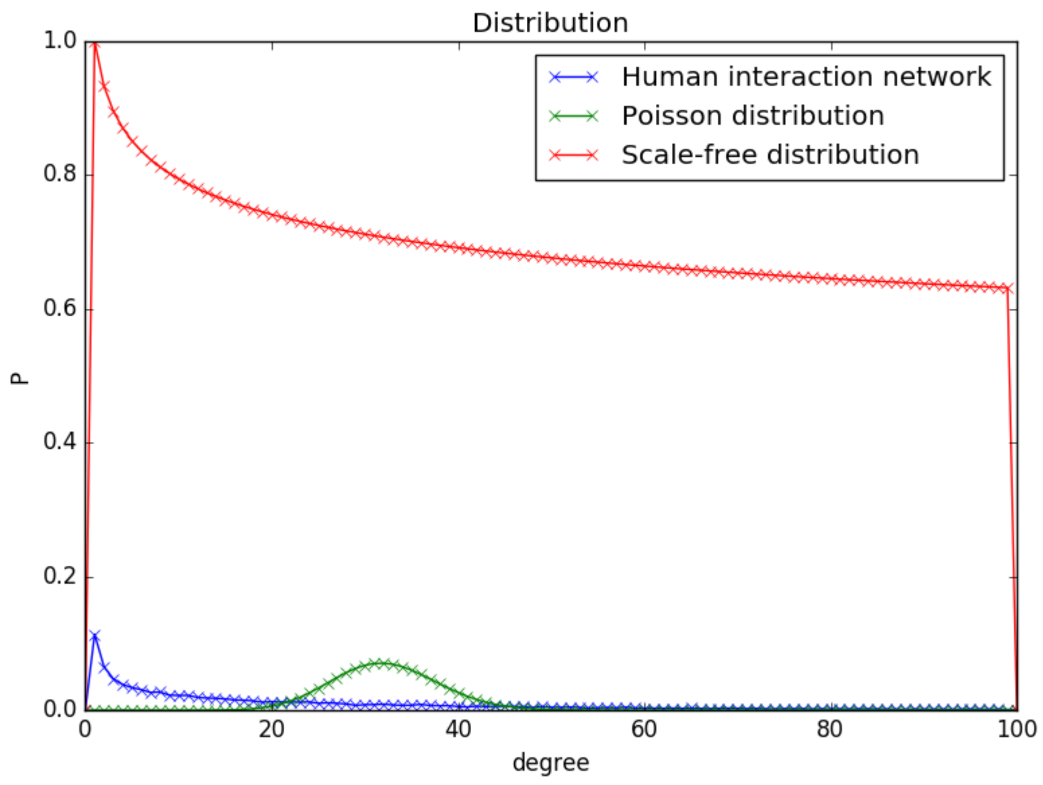

Figure 3: Comparison of empirical to theoretical network