Bioinformatics III

Fifth Assignment

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

May 25, 2018

Exercise 5.1: Cliques and Network Evolution

(a) Listing 1 shows source code.

Listing 1: Listing of source code

```
import random
from os.path import exists
from AbstractNetwork import AbstractNetwork
import matplotlib.pyplot as plt


class CliqueNetwork(AbstractNetwork):
    """
    function that computes the number of cliques of sizes 3, 4 and 5 in a network
    """
    def __createNetwork__(self, filename, dummy):
        """
        Create a network from a file
        """
        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 findCliques(self):
        """
        function that computes the number of cliques of sizes 3, 4 and 5 in a network.
        :return: list of cliques
        """
        cliques = []
        for node1 in self.nodes.values():
            for node2 in node1.nodelist:
                candidates = list()
                candidates.append(node1)
                candidates.append(node2)
                clique = set(self.extendClique(candidates, 3))
                #only consider cliques of size 3-5
                if clique not in cliques and len(clique) > 2:
                    cliques.append(clique)
        return cliques
    def extendClique(self, candidates, depth):
        """
        try to extend clique to larger cliques
        :param candidates: smaller clique
        :param depth: clique size; maximal 5
        :return: largest clique possible
        """
        if depth <= 5:
            for nextnode in candidates[0].nodelist:
                if nextnode not in candidates:
                    #check if node is connected to all nodes in clique
                    if all(nextnode.hasLinkTo(x) for x in candidates):
                        candidates.append(nextnode)
                        self.extendClique(candidates, depth+1)
                    else:
                        return candidates
            return candidates
        else:
            return candidates

    def evolve(self, t):
        """
        function that takes a parameter t representing the number of time steps, as well as a r
        step, randomly insert or delete one edge in the network
        :param t: number of time steps
        """
        cliques = list()
        for c in range(3):
            cliques.append(t * [0])
        for i in range(t):
            clique = cn.findCliques()
            r = random.random()
            node1id = random.choice(list(self.nodes))
            node1 = self.nodes[node1id]
            #decide if edge gets removed or new edge added
            if r <= 0.5:
                #try to find two nodes without edge
                while True:
                    node2id = random.choice(list(self.nodes))
                    node2 = self.nodes[node2id]
                    if not node1.hasLinkTo(node2):
                        node1.addLinkTo(node2)
                        node2.addLinkTo(node1)
                        break
            else:
                #search for edge
                while len(node1.nodelist) == 0:
                    node1id = random.choice(list(self.nodes))
                    node1 = self.nodes[node1id]
                node2 = random.choice(node1.nodelist)
                node1.removeLinkTo(node2)
            #plot number of cliques of size 3, 4 and 5 at the beginning and after each time ste
            #  with t = 100
            cliques = self.plot(clique, i, cliques)
        for index, item in enumerate(cliques):
            plt.plot(range(len(item)), item, marker='x')
            plt.xlabel('t')
            plt.ylabel('amount_of_cliques')
            plt.title('Evolving_Networks')
            plt.legend('345')
            plt.tight_layout()
        plt.show()

    def printClique(self, clique):
        """
        print cliques to check result
```

```
            :param clique: clique to print
            """
            for line in clique:
                s = ''
                for c in line:
                    s += str(c)
                print(s)

        def plot(self, clique, t, cliques):
            """
            count amount of cliques of size i at time step t
            :param clique: cliques in current time step
            :param t: time step
            :param cliques: matrix to store count
            :return: matrix with counts
            """
            for c in clique:
                cliques[len(c)-3][t] = cliques[len(c)-3][t] + 1
            return cliques

        def numLinks(self):
            summ = 0
            for n in self.nodes:
                summ = summ + len(self.getNode(n).nodelist)
            return summ / 2
```

(b) Listing 1 shows source code.

(c) Listing 1 shows source code.

(d) The number of cliques stays approximately the same independent of the time step because approximately the same amount of edges are added which are removed as shown in figure 1, table 4 and table 4. The amount of cliques of size 3 increases rather than the one of larger cliques because a less edges added may result in a small clique.

clique size	number of cliques before evolution	number of cliques after evolution
3	294	300
4	33	33
5	3	3

Table 1: Number of cliques of size 3, 4 and 5 at the beginning and after letting it evolve for 100 time steps

clique size	number of cliques before evolution	number of cliques after evolution
3	294	327
4	33	27
5	3	3

Table 2: Number of cliques of size 3, 4 and 5 at the beginning and after letting it evolve for 1000 time steps

(e) The goal of randomising networks this way is to create random permutations of a network. In this way the behaviour of similar networks may be studied and the networks quality may be rated.
Listing 2 shows source code.

Listing 2: Listing of source code

```
from AbstractNetwork import AbstractNetwork
import random
```

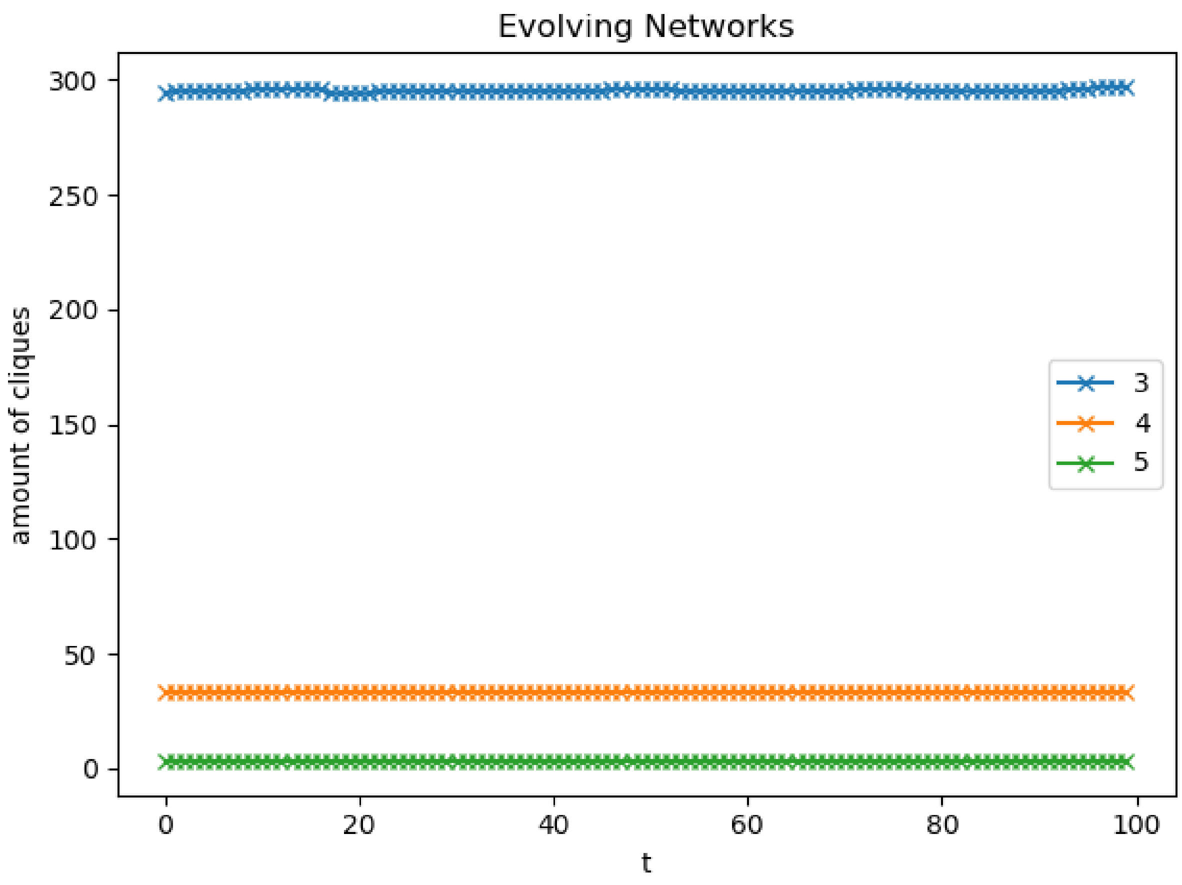

Figure 1: Number of cliques of size 3, 4 and 5 at the beginning and after each time step as a function of time with t = 100

```
from CliqueNetwork import CliqueNetwork


class RandomizedNetwork(AbstractNetwork):
    """
    function that takes a network with m edges and returns a randomised version of that network
    """
    def __createNetwork__(self, network, m):
        """
        for 2m iterations, randomly select two edges e1 = (n1; n2) and e2 = (n3; n4) from the n
        such that the start and end nodes are swapped
        :param network: network
        :param m: number of edges
        """
        self.nodes = network.nodes
        for i in range(2*m):
            node1 = self.chooseNode()
            node2 = random.choice(node1.nodelist)
            node3 = self.chooseNode()
            node4 = random.choice(node3.nodelist)
            node1.addLinkTo(node4)
            node3.addLinkTo(node2)
            node1.removeLinkTo(node2)
            node3.removeLinkTo(node4)

    def chooseNode(self):
        """
        choose random node with edges
        :return: random node with at least one edge
        """
        nodeid = random.choice(list(self.nodes))
        node = self.nodes[nodeid]
        if len(node.nodelist) != 0:
            return node
        else:
            self.chooseNode()


cn = RandomizedNetwork(CliqueNetwork("test.tsv", "dummy"), 10)
```

(f) Assuming a p-value of 0.05, none of the clique sizes were significantly enriched, see table 6. Listing 3 shows source code.

Listing 3: Listing of source code

```
import random
from CliqueNetwork import CliqueNetwork


class MotifNetworks:
    """
    class that takes a parameter n and a network and computes if cliques of size 3, 4 and 5 are
     enriched in that network
    """
    network = CliqueNetwork("", "")
    clique = list()

    def __init__(self, network, n):
        self.network = network
        #compute number of cliques in original network
        self.clique = self.network.findCliques()
        #perform n switches
        cliques = self.__createNetwork__(n)
        c_i = dict()
        #initilize dict
        for i in range(3, 6):
                c_i[i] = 0
        #count cliques of size i
        for i in self.clique:
            c_i[len(i)] = c_i.get(len(i), 0) + 1
        # count cliques of size of at least i
        for i in range(4, 2, -1):
                c_i[i] += c_i[i+1]
        #compute p_i
        extremes = list()
        for key, item in c_i.items():
            extremes.append((key, self.count(cliques, key, item)))
        print(extremes)


    def __createNetwork__(self, n):
        """
        for 2m iterations, randomly select two edges e1 = (n1; n2) and e2 = (n3; n4) from the 
        such that the start and end nodes are swapped
        :param n: number of swaps
        """
        cliques = list()
        for i in range(n):
            node1 = self.chooseNode()
            node2 = random.choice(node1.nodelist)
            node3 = self.chooseNode()
            node4 = random.choice(node3.nodelist)
            node1.addLinkTo(node4)
            node3.addLinkTo(node2)
            node1.removeLinkTo(node2)
            node3.removeLinkTo(node4)
            clique = self.network.findCliques()
            cliques.append(clique)
        return cliques

    def count(self, cliques, i, c_i):
        """
        count cliques of size of at least i
        :param cliques: randomized cliques
        :param i: clique size
        :param c_i: number of cliques with clique size of at least i in original network
        :return: number of networks with at least as many cliques of size of at least i over n
                networks
        """
        extremes = 0
        for network in cliques:
            count = 0
            for clique in network:
                if len(clique) >= i:
                    count += 1
            if count >= c_i:
                extremes += 1
        extremes /= float(len(cliques))
        return extremes

    def chooseNode(self):
        """
        choose random node with edges
        :return: random node with at least one edge
        """
        nodeid = random.choice(list(self.network.nodes))
        node = self.network.nodes[nodeid]
        if len(node.nodelist) != 0:
            return node
        else:
            self.chooseNode()


cn = CliqueNetwork("rat_network.tsv", "dummy")
```

```
    MotifNetworks(cn, 100)
```

clique size i	p_i
3	0.38
4	0.27
5	1.0

Table 3: Number of randomised networks in which the number of cliques is at least as high as in the original network divided by the number of randomised networks for each clique size

Exercise 5.2: Annotations in Protein Protein Interaction Networks

(a) Implemented in `PPIGONetwork.py`, see Listing 4

(b) Required information provided in table 4

Table 4: Reults for exercise 5.2 b

Task	Result
total number of proteins and interactions in the network	17087
total number of unique annotations in the network	275472
total number and percentage of proteins without any annotation	2334 (13.6595%)
smallest number of annotations per protein	0
average number of annotations per protein	7.013
highest number of annotations per protein	184
smallest number of associated proteins per annotatio	1
average number of associated proteins per annotatio	10.302
highest number of associated proteins per annotatio	1519

(c) In table 5, we see the five most and least annotated GO terms. We see that the most annotated GO terms are all involved in crucial biological processes for cell viability. In this way, we have important transcription factors, which regulate the different expression levels. Additonally, we have one GO term encoding signal transduction, which is also important for cell communication, appoptosis, etc. Further, the GO terms are relatively unspecific, so they do not describe just one specific process, - this fewer information content leads to a high number of annotations . Therefore it is no surprise that these GO terms are high annotated. In contrast, the least annotated GO terms describe various, precise biological processes. Therefore they can only be assigned to a specific subset of human proteins. Therefore we see only a few annotations of them. So they are less common but have a higher information content than the most annotated proteins GO terms.

(d) Not implemented

(e) Not implemented

(f) Listing 4 shows source code.

Listing 4: Listing of source code

```
from CliqueNetwork import CliqueNetwork
from os.path import exists
import sys

class Protein:
    """
    Protein class: Storage for protein specific informations
    obtained by the network, uniprot and go-term-file
```

Table 5: Reults for exercise 5.2 b

Task	Result	
	Most Common	
GO term	Occurrence	Biological Process
GO:0006351	1519	transcription, DNA-templated
GO:0045944	1001	positive regulation of transcription by RNA polymerase II
GO:0007165	980	signal transduction
GO:0006357	937	regulation of transcription by RNA polymerase II
GO:0006355	740	regulation of transcription, DNA-templated
	Least Common	
GO term	Occurrence	Biological Process
GO:0000003	1	reproduction
GO:0000011	1	vacuole inheritance
GO:0000032	1	cell wall mannoprotein biosynthetic process
GO:0000053	1	argininosuccinate metabolic process
GO:0000097	1	sulfur amino acid biosynthetic process

```
    """
    def __init__(self, name):
        # storage for protein name (from PPI)
        self.name = name
        # storage for accession number from Go file
        self.accession = ""
        # storage for alternative names (uniprot)
        self.alternatives = set([name])
        # storage for go terms from go file
        self.GOs = set()

    # Getter
    def getName(self):
        return self.name
    def getAccessionNumber(self):
        return self.accession
    def getAlternatives(self):
        return self.alternatives
    def getGOTerms(self):
        return self.GOs

    # Setter
    def setName(self, name):
        self.name = name
    def setAccessionNumber(self, accession):
        self.accession = accession
    def addAlternatives(self, alternatives):
        for a in alternatives:
            self.alternatives.add(a)
    def addGOTerm(self, go):
        self.GOs.add(go)

    # terminal visualization
    def show(self):
        print "Name:\t\t" + self.name
        print "Accession Number:\t" + self.accession
        print "Alternative Names:"
        counter = 1
        for i in self.alternatives:
            print "\t" + str(counter) + "\t" + i
            counter = counter + 1
        print "GO Term Annotations:"
        counter = 1
        for i in self.GOs:
            print "\t" + str(counter) + "\t" + i
            counter = counter + 1


class Mapping:
    """
    Mapping class: Stores protein objects and provides
    further functionality
    """
    def __init__(self):
        """
        key: protein name = Protein
        """
        self.mapping = dict()

    def existsAccession(self, accession):
        """
        Returns true if the mapping contains a protein with
        the given accession number
        """
        for name in self.mapping.keys():
            if self.mapping[name].getAccessionNumber() == accession:
                return True
        return False

    def existsName(self, name):
        """
        Returns true if the mapping contains a protein with
        the given name
        """
        if name in self.mapping.keys():
            return True
        else:
            return False

    def getMainName(self, variant):
        """
        Return the name of a protein for a given alternative name
        """
        if self.existsName(variant):
            return variant
        for i in self.mapping.keys():
            if variant in self.mapping[i].getAlternatives():
                return i
        return variant

    def addProtein(self, name):
        """
        Expand mapping by one protein with the given name
        """
        self.mapping[name] = Protein(name)

    def addAlternatives(self, name, alternatives):
        """
        Add an alternative name to the protein with the given name
        Return without action if the protein name does not exist
        """
        if self.existsName(name):
            self.mapping[name].addAlternatives(alternatives)

    def addGOTerm(self, name, goterm):
        """
        Add a go term to the protein with the given name.
        Return without action if the protein name does not exist
        """
        if self.existsName(name):
            self.mapping[name].addGOTerm(goterm)
    def addAccession(self, name, accession):
        """
        Add an accession number to the protein with the given name
        Return without action if the protein name does not exist
        """
        if self.existsName(name):
            self.mapping[name].setAccessionNumber(accession)

    def show(self):
        """
        Write proteines into the terminal
        """
        counter = 1
        for i in self.mapping.keys():
            print str(counter) + ".\t---------------------------------------------"
            self.mapping[i].show()
            counter = counter + 1

    def getKeys(self):
        """
        Return mapping keys (in the end, these are the protein names)
        """
        return self.mapping.keys

    # Exercise specific functions

    def sumNoAnnotation(self):
        """
        Count proteins without go annotations
        """
        summ = 0
        for i in self.mapping.keys():
            if len(self.mapping[i].getGOTerms()) == 0:
                summ += 1
        return summ

    def rawDistribution(self):
        """
        Compute the minimal, average and maximal number
        of go annotations per gene
        """
        values = [sys.maxint,0,0]
        for i in self.mapping.keys():
            annotations = len(self.mapping[i].getGOTerms())
            if annotations < values[0]:
                values[0] = annotations
            if annotations > values[2]:
                values[2] = annotations
            values[1] = values[1] + annotations
        values[1] = float(values[1])/float(len(self.mapping.keys()))
        return values

    def rawReverseDistribution(self):
        """
        Compute the minimal, average and maximal number
        of proteins per go annotations
        """
        # initialize dict (key: GO term, value: occurrence)
        # store it as field of the class to allow later access
        self.revsum = dict()
        for i in self.mapping.keys():
            terms = self.mapping[i].getGOTerms()
            for j in terms:
                self.revsum[j] = 0
        # count occurrences
        for i in self.mapping.keys():
            terms = self.mapping[i].getGOTerms()
            for j in terms:
                self.revsum[j] += + 1

        # evaluate minimum, average and maximum
        values = [sys.maxint,0,0]
        for i in self.revsum.keys():
            temp = self.revsum[i]
            if temp < values[0]:
                values[0] = temp
            if temp > values[2]:
                values[2] = temp
            values[1] = values[1] + temp
        values[1] = float(values[1])/float(len(self.revsum.keys()))
        return values

    def extendedNumber(self, length, number):
        """
        Append zeros in front of a string, allows a lexicographical
        sorting of different sized numbers
        """
        while len(number) < length:
            number = "0" + number
        return number

    def nMostFewestAnnotations(self, n):
        """
        Returns the n most and n fewest annotated GO terms
        """
        # create a dict, where the keys are a combination of occurrences
        # and GO terms (as string - allows easy sorting)
        sorter = dict()
        for i in self.revsum.keys():
            key = self.extendedNumber(10, str(self.revsum[i])) + i
            sorter[key] = (i, self.revsum[i])

        # sort the keys of the sorter dict in lexicographically increaing
        # order, at the same time the most occurring terms come to the end
        helping = []
        for i in sorter.keys():
            helping.append(i)
        helping.sort()

        # extract the n most and fewest common GO annotations
        high = []
        low = []
        for i in helping[0:n]:
            low.append(sorter[i])
        for i in helping[(len(helping)-n):len(helping)]:
            high.append(sorter[i])
        return [high, low]


class PPIGONetwork:
    """
    Performs part a b c of exercise 5.2
    """
    def __init__(self, ppi_file, uniprot_file, go_file):
        # read in ppi network
        print "1. Create Network"
        self.ppi = CliqueNetwork(ppi_file, "")
        # create mapping of protein informations
        print "2. Initialize Mapping"
        self.mapping = Mapping()
        self.initializeMapping()
        print "3. Assign Alternative Names"
        self.updateMappingNames(uniprot_file)
        print "4. Assign Go terms"
        self.assignGOTerms(go_file)
        print "Initialization completed"

    def initializeMapping(self):
        """
        Transfere network proteins into mapping
        """
        for n in self.ppi.nodes:
            self.mapping.addProtein(n)

    def updateMappingNames(self, uniprot_file):
        """
        Read in uniprot file to extend mapping with alternative
        protein names
        """
        if exists(uniprot_file):
            with open(uniprot_file) as openfile:
                for line in openfile:
                    # get entries of a line as list
                    content = line[0:(len(line)-1)].split("\t")
                    if len(content) >= 4:
                        variants = content[4].split(" ")
                        for temp in variants:
                            self.mapping.addAlternatives(temp, variants)
        else:
            print(filename, "does not exist")

    def assignGOTerms(self, go_file):
        """
        Extend mapping with GO terms of GO term file
        Add accession number to proteins
        """
        if exists(go_file):
            # "with" closes the file again after reading
            with open(go_file) as openfile:
                for line in openfile:
                    # get entries of a line as list
                    content = line[0:(len(line)-1)].split("\t")
                    #content = [x for x in content if x != ""]
                    if content[0] == "UniProtKB" and len(content) > 8:
                        if content[8] == "P":
                            temp = self.mapping.getMainName(content[2])
                            self.mapping.addGOTerm(temp, content[4])
                            self.mapping.addAccession(temp, content[1])

    def size(self):
        return self.ppi.size()

    def overview(self):
        """
        Output information required for subtasks a b c
        """
        print "------------------------- Overview -------------------------"
        print "Total number of proteins:\t\t\t" + str(self.ppi.size())
        print "Total number of protein interactions:\t\t" + str(self.ppi.numLinks())
        no = self.mapping.sumNoAnnotation()
        percent = (float(no)/float(self.ppi.size()))*100.0
        print "Number of proteines without GO annotation:\t" + str(no) + " (" + str(percent) +
        distr = self.mapping.rawDistribution()
        print "Smallest number of GO annotations per protein\t" + str(distr[0])
        print "Average number of GO annotations per protein\t" + str(distr[1])
        print "Highest number of GO annotations per protein\t" + str(distr[2])
        distr = self.mapping.rawReverseDistribution()
        print "Smallest number of proteins per GO annotations\t" + str(distr[0])
        print "Average number of proteins per GO annotations\t" + str(distr[1])
        print "Highest number of proteins per GO annotations\t" + str(distr[2])
        temp = self.mapping.nMostFewestAnnotations(5)
        print "5 most annotated GO terms:\t" + str(temp[0])
        print "5 fewest annotated GO terms:\t" + str(temp[1])
```

Program output for human network with time measurement:

```
Start program
1.  Create Network
2.  Initialize Mapping
3.  Assign Alternative Names
4.  Assign Go terms
Initialization completed
--------------------- Overview ---------------------
Total number of proteins:  17087
Total number of protein interactions:  275472
Number of proteines without GO annotation:  2334 (13.6595072277%)
Smallest number of GO annotations per protein 0
Average number of GO annotations per protein 7.01293380933
Highest number of GO annotations per protein 184
Smallest number of proteins per GO annotations 1
Average number of proteins per GO annotations 10.3017537827
Highest number of proteins per GO annotations 1519
5 most annotated GO terms:  [('GO:0006355', 740), ('GO:0006357', 937), ('GO:0007165',
980), ('GO:0045944', 1001), ('GO:0006351', 1519)]
5 fewest annotated GO terms:  [('GO:0000003', 1), ('GO:0000011', 1), ('GO:0000032',
1), ('GO:0000053', 1), ('GO:0000097', 1)]

real 5m56,373s
user 5m53,229s
sys 0m0,304s
```