Bioinformatics III

Fifth Assignment

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

May 23, 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
```

```
45
    def extendClique(self, candidates, depth):
        """
        try to extend clique to larger cliques
        :param candidates: smaller clique
50      :param depth: clique size; maximal 5
        :return: largest clique possible
        """
        if depth <= 5:
            for nextnode in candidates[0].nodelist:
55              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)
60                  else:
                        return candidates
            return candidates
        else:
            return candidates
65
    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
70      :param t: number of time steps
        """
        cliques = list()
        for c in range(3):
            cliques.append(t * [0])
75      for i in range(t):
            clique = cn.findCliques()
            r = random.random()
            node1id = random.choice(list(self.nodes))
            node1 = self.nodes[node1id]
80          #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))
85                  node2 = self.nodes[node2id]
                    if not node1.hasLinkTo(node2):
                        node1.addLinkTo(node2)
                        node2.addLinkTo(node1)
                        break
90          else:
                #search for edge
                while len(node1.nodelist) == 0:
                    node1id = random.choice(list(self.nodes))
                    node1 = self.nodes[node1id]
95              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)
100     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')
105         plt.legend('345')
            plt.tight_layout()
        plt.show()

    def printClique(self, clique):
110     """
        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


cn = CliqueNetwork("rat_network.tsv", "dummy")
clique = cn.findCliques()
cn.evolve(1000)
clique = cn.findCliques()
```

(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
```

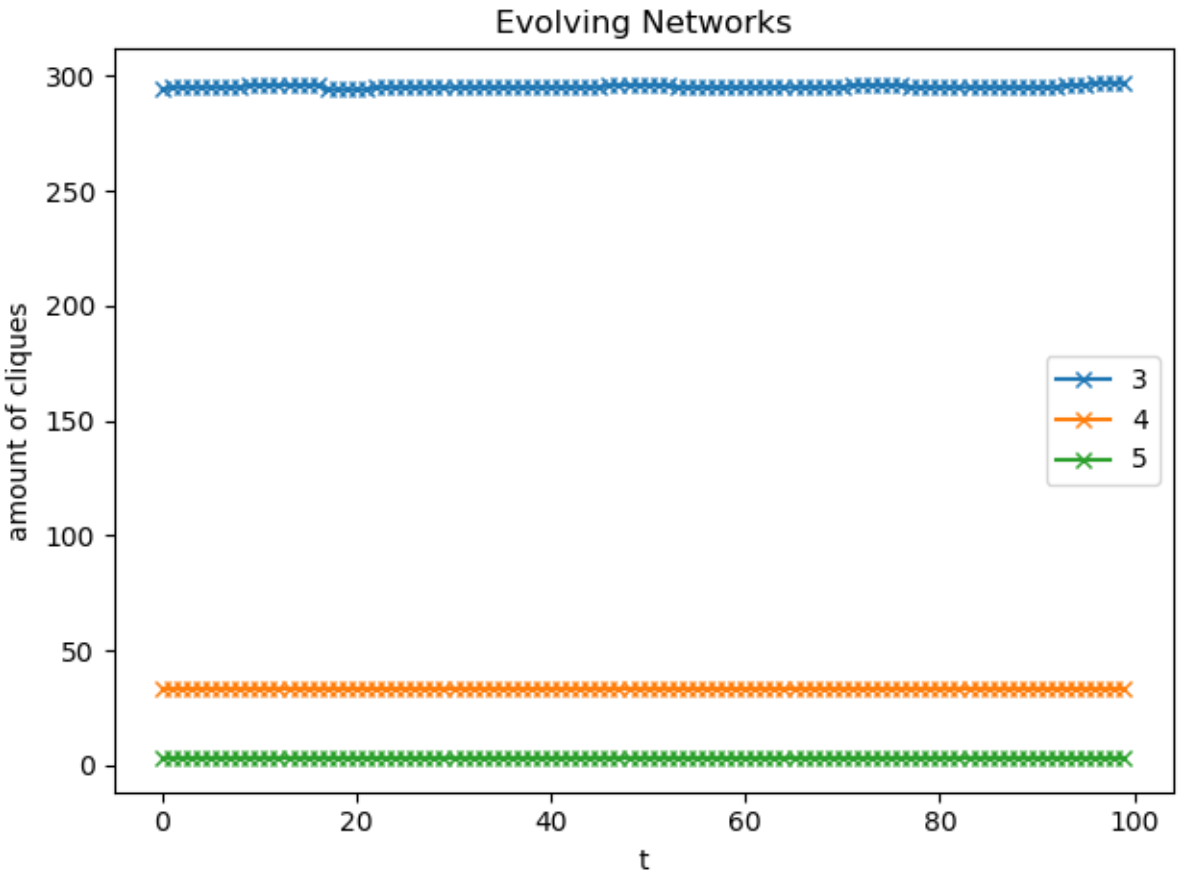

Figure 1: Number of cliques of size 3, 4 and 5 at the beginning and after each time step as a function of time with t = 100

```
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 n
        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 nu
                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)

(b)

(c)

(d)

(e)