

# 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

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

5
  class CliqueNetwork(AbstractNetwork):
      """
      function that computes the number of cliques of sizes 3, 4 and 5 in a network
      """
10  def __createNetwork__(self, filename, dummy):
      """
      Create a network from a file
      """
      if exists(filename):
15          # "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")
20                  # and store them
                  if len(content) == 2:
                      n1 = self.getNode(content[0])
                      n2 = self.getNode(content[1])
                      n1.addLinkTo(n2)
                      n2.addLinkTo(n1)
25              else:
                  print(filename, "does_not_exist")

  def findCliques(self):
30      """
      function that computes the number of cliques of sizes 3, 4 and 5 in a network.
      :return: list of cliques
      """
      cliques = []
35      for node1 in self.nodes.values():
          for node2 in node1.nodelist:
              candidates = list()
              candidates.append(node1)
              candidates.append(node2)
40              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:
            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 n
    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
            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 step
            # 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:
115         s = ''
            for c in line:
                s += str(c)
            print(s)

120     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
125         :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
130         return cliques

    def numLinks(self):
        summ = 0
        for n in self.nodes:
135             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

```

0 from AbstractNetwork import AbstractNetwork
import random

```

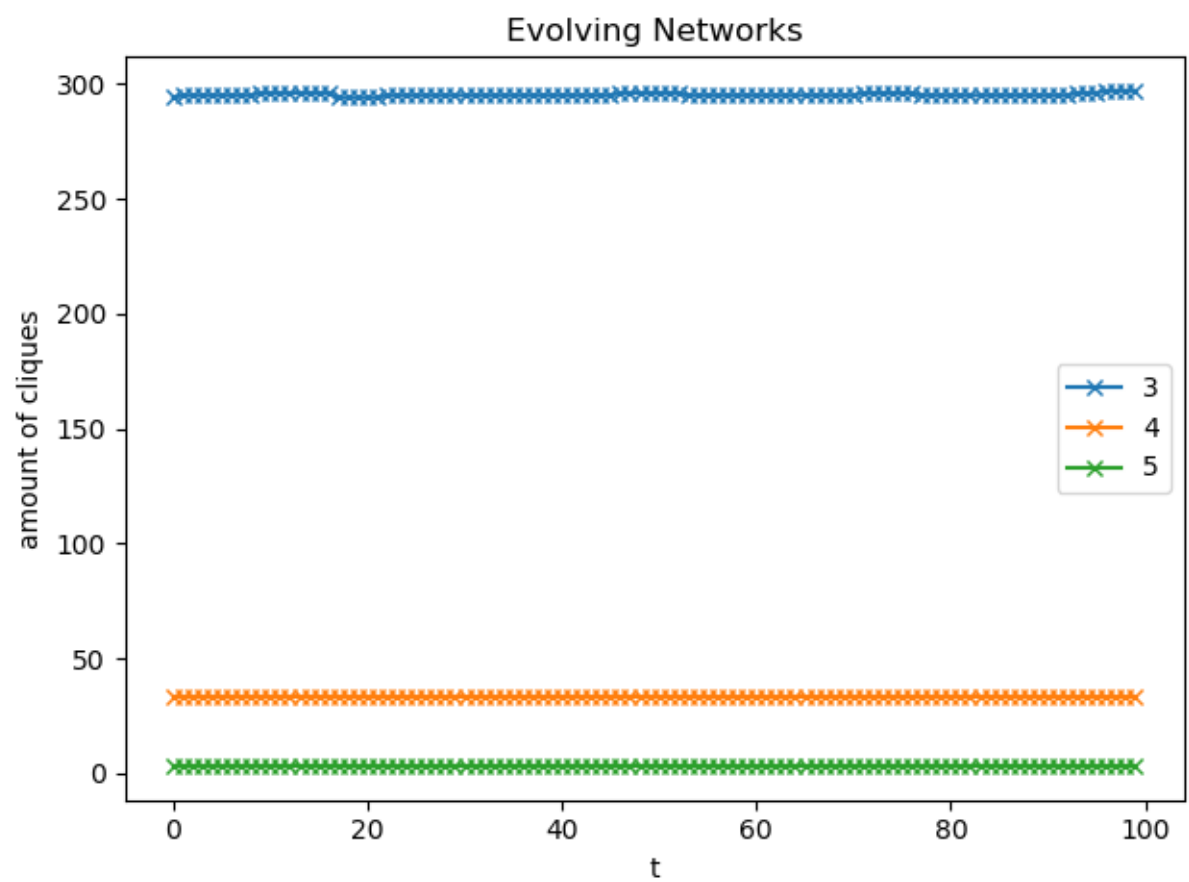


Figure 1: Number of cliques of size 3, 4 and 5 at the beginning and after each time step as a function of time with  $t = 100$

```

from CliqueNetwork import CliqueNetwork

5
class RandomizedNetwork(AbstractNetwork):
    """
    function that takes a network with m edges and returns a randomised version of that network
    """
10    def __createNetwork__(self, network, m):
        """
        for 2m iterations, randomly select two edges e1 = (n1; n2) and e2 = (n3; n4) from the network
        such that the start and end nodes are swapped
        :param network: network
15        :param m: number of edges
        """
        self.nodes = network.nodes
        for i in range(2*m):
            node1 = self.chooseNode()
20            node2 = random.choice(node1.nodelist)
            node3 = self.chooseNode()
            node4 = random.choice(node3.nodelist)
            node1.addLinkTo(node4)
            node3.addLinkTo(node2)
25            node1.removeLinkTo(node2)
            node3.removeLinkTo(node4)

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

40
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

```

0 import random
from CliqueNetwork import CliqueNetwork

class MotifNetworks:
5    """
    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("", "")
10    clique = list()

    def __init__(self, network, n):
        self.network = network
        #compute number of cliques in original network
15        self.clique = self.network.findCliques()
        #perform n switches
        cliques = self.__createNetwork__(n)
        c_i = dict()
        #initilize dict
20        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
25    # 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()
30    for key, item in c_i.items():
        extremes.append((key, self.count(cliques, key, item)))
        print(extremes)

35    def __createNetwork__(self, n):
        """
        for 2m iterations, randomly select two edges e1 = (n1; n2) and e2 = (n3; n4) from the network
        such that the start and end nodes are swapped
        :param n: number of swaps
        """
40        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)
50        clique = self.network.findCliques()
            cliques.append(clique)
        return cliques

55    def count(self, cliques, i, c_i):
        """
        count cliques of size of at least i
        :param cliques: randomized cliques
        :param i: clique size
60        :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
65        for network in cliques:
            count = 0
            for clique in network:
                if len(clique) >= i:
                    count += 1
70            if count >= c_i:
                extremes += 1
        extremes /= float(len(cliques))
        return extremes

75    def chooseNode(self):
        """
        choose random node with edges
        :return: random node with at least one edge
        """
80        nodeid = random.choice(list(self.network.nodes))
        node = self.network.nodes[nodeid]
        if len(node.nodelist) != 0:
            return node
        else:
85            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: Results for exercise 5.2 b			
Task	Human	Chicken	Pig
total number of proteins and interactions in the network	17087	281	51
total number of unique annotations in the network	275472	300	50
total number and percentage of proteins without any annotation	2334 (13.6595%)	44 (15.658%)	13(25.49%)
smallest number of annotations per protein	0	0	0
average number of annotations per protein	7.013	7.135	5.294
highest number of annotations per protein	184	77	40
smallest number of associated proteins per annotation	1	1	1
average number of associated proteins per annotation	10.302	1.558	1.134
highest number of associated proteins per annotation	1519	27	5

- (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. Additionally, we have one GO term encoding signal transduction, which is also important for cell communication, apoptosis, 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

```

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

  class Protein:
    """
5

```

Table 5: Results 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

```

Protein class: Storage for protein specific informations
obtained by the network, uniprot and go-term-file
"""
def __init__(self, name):
10     # storage for protein name (from PPI)
    self.name = name
    # storage for accession number from Go file
    self.accession = ""
    # storage for alternative names (uniprot)
15     self.alternatives = set([name])
    # storage for go terms from go file
    self.GOs = set()

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

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

40     # terminal visualization
    def show(self):
        print "Name:\t\t" + self.name
        print "Accession_Number:\t" + self.accession
        print "Alternative_Names:"
45         counter = 1
        for i in self.alternatives:
            print "\t" + str(counter) + "\t" + i
            counter = counter + 1
        print "GO_Term_Annotations:"

```



```
50         counter = 1
           for i in self.GOs:
               print "\t" + str(counter) + "\t" + i
               counter = counter + 1

55
class Mapping:
    """
    Mapping class: Stores protein objects and provides
    further functionality
    """
60     def __init__(self):
        """
        key: protein name = Protein
        """
65         self.mapping = dict()

    def existsAccession(self, accession):
        """
        Returns true if the mapping contains a protein with
        the given accession number
70         """
        for name in self.mapping.keys():
            if self.mapping[name].getAccessionNumber() == accession:
                return True
75         return False

    def existsName(self, name):
        """
        Returns true if the mapping contains a protein with
        the given name
80         """
        if name in self.mapping.keys():
            return True
        else:
85             return False

    def getMainName(self, variant):
        """
        Return the name of a protein for a given alternative name
90         """
        if self.existsName(variant):
            return variant
        for i in self.mapping.keys():
            if variant in self.mapping[i].getAlternatives():
95                 return i
        return variant

    def addProtein(self, name):
        """
        Expand mapping by one protein with the given name
100        """
        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
105        """
        if self.existsName(name):
            self.mapping[name].addAlternatives(alternatives)
110

    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
115        """
```

```
        if self.existsName(name):
            self.mapping[name].addGOTerm(goterm)

120 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
    """
125     if self.existsName(name):
        self.mapping[name].setAccessionNumber(accession)

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

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

# Exercise specific functions
145
def sumNoAnnotation(self):
    """
    Count proteins without go annotations
    """
150     summ = 0
    for i in self.mapping.keys():
        if len(self.mapping[i].getGOTerms()) == 0:
            summ += 1
    return summ

155
def rawDistribution(self):
    """
    Compute the minimal, average and maximal number
    of go annotations per gene
160     """
    values = [sys.maxint, 0, 0]
    for i in self.mapping.keys():
        annotations = len(self.mapping[i].getGOTerms())
        if annotations < values[0]:
165             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()))
170     return values

def rawReverseDistribution(self):
    """
    Compute the minimal, average and maximal number
175     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()
180     for i in self.mapping.keys():
        terms = self.mapping[i].getGOTerms()
        for j in terms:
            self.revsum[j] = 0
```

```
185         # count occurrences
        for i in self.mapping.keys():
            terms = self.mapping[i].getGOTerms()
            for j in terms:
                self.revsum[j] += 1

190         # evaluate minimum, average and maximum
        values = [sys.maxint, 0, 0]
        for i in self.revsum.keys():
            temp = self.revsum[i]
            if temp < values[0]:
195                 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()))
200        return values

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

210    def nMostFewestAnnotations(self, n):
        """
        Returns the n most and n fewest annotated GO terms
        """
215        # 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
220            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 = []
225        for i in sorter.keys():
            helping.append(i)
        helping.sort()

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

class PPIGONetwork:
240    """
    Performs part a b c of exercise 5.2
    """
    def __init__(self, ppi_file, uniprot_file, go_file):
        # read in ppi network
245        print "1. Create Network"
        self.ppi = CliqueNetwork(ppi_file, "")
        # create mapping of protein informations
        print "2. Initialize Mapping"
        self.mapping = Mapping()
250        self.initializeMapping()
```

```

    print "3. Assign Alternative Names"
    self.updateMappingNames( uniprot_file )
    print "4. Assign GO terms"
    self.assignGOTerms( go_file )
255    print "Initialization completed"

def initializeMapping( self ):
    """
    Transfere network proteins into mapping
    """
260    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 ):
270        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:
275                    variants = content[4].split("_")
                    for temp in variants:
                        self.mapping.addAlternatives( temp , variants )
            else:
                print( filename , "does_not_exist" )
280

def assignGOTerms( self , go_file ):
    """
    Extend mapping with GO terms of GO term file
    Add accession number to proteins
    """
285    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
290                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":
295                        temp = self.mapping.getMainName( content[2] )
                        self.mapping.addGOTerm( temp , content[4] )
                        self.mapping.addAccession( temp , content[1] )

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

def overview( self ):
    """
    Output information required for subtasks a b c
    """
305    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
310    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] )
315    distr = self.mapping.rawReverseDistribution()
    print "Smallest_number_of_proteins_per_GO_annotations\t" + str( distr[0] )
```

```
320      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 proteins 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

Program output for chicken 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: 281

Total number of protein interactions: 300

Number of proteins without GO annotation: 44 (15.6583629893%)

Smallest number of GO annotations per protein 0

Average number of GO annotations per protein 7.13523131673

Highest number of GO annotations per protein 77

Smallest number of proteins per GO annotations 1

Average number of proteins per GO annotations 1.55788655789

Highest number of proteins per GO annotations 27

5 most annotated GO terms: [('GO:0000122', 15), ('GO:0006281', 18), ('GO:0006355', 18), ('GO:0045944', 21), ('GO:0006351', 27)]

5 fewest annotated GO terms: [('GO:0000027', 1), ('GO:0000028', 1), ('GO:0000054', 1), ('GO:0000077', 1), ('GO:0000082', 1)]

real 0m4,789s

user 0m4,716s

sys 0m0,024s

Program output for pig 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: 51

Total number of protein interactions: 50

Number of proteins without GO annotation: 13 (25.4901960784%)

Smallest number of GO annotations per protein 0

Average number of GO annotations per protein 5.29411764706

Highest number of GO annotations per protein 40

Smallest number of proteins per GO annotations 1

Average number of proteins per GO annotations 1.13445378151

Highest number of proteins per GO annotations 5

5 most annotated GO terms: [('GO:0045944', 3), ('GO:0055114', 3), ('GO:0006886', 4), ('GO:0007338', 4), ('GO:0006511', 5)]

5 fewest annotated GO terms: [('GO:0000077', 1), ('GO:0000132', 1), ('GO:0000187', 1), ('GO:0000226', 1), ('GO:0001542', 1)]

real 0m1,639s

user 0m1,539s

sys 0m0,044s