Bioinformatics III

Sixth Assignment

Max Jakob (2549155)
Carolin Mayer (2552320)

June 5, 2018

Exercise 6.1: Boolean Networks

a) The propagation matrix for the given graph is shown in Table 1
b) It make sense to stop the propagation as soon as we enter a loop or enter a state where non of the nodes are active. A loop means that a state is entered, which already has been enter. If we wouldn't stop there, we indefinitely run in this loop.
The sequences according to the states 1,4,21,33:
1: [1, 3, 7, 23, 55, 63, 13, 1]
4: [4, 18, 36, 26, 4]
21:[21, 51, 47, 13, 1, 3, 7, 23, 55, 63, 13]
33: [33, 11, 5, 19, 39, 31, 5]
c)

attractor	period length	basins	coverage
1	7	[1, 9, 25, 41, 29, 45, 57, 61]	0.125
3	5	[3]	0.15625
4	4	[2, 4, 10, 14, 38, 30]	0.09375
5	4	[5, 17, 33, 11, 35, 37, 15, 27]	0.125
7	7	[7]	0.015625
39	4	[39]	0.015625
13	7	[13, 21, 49, 43, 51, 53, 47, 59]	0.125
18	4	[18]	0.015625
19	4	[19]	0.015625
23	7	[23]	0.015625
36	4	[36]	0.015625
26	4	[26]	0.015625
63	7	[63]	0.015625
55	7	[55]	0.015625
31	4	[31]	0.015625

d) In the following the occurrences of each letter in orbit is showed relatively to the length of the orbit in the scheme attractor: [A,B,C,D,E,F]
As special gene for example wold be A, as it is rather always on, or always off. Furthermore, we can say that the rate of B is always equal to the rate of C, and the rate of D always equal to E.
1: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
3: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
4: [0.0, 0.5, 0.5, 0.25, 0.5, 0.25]
5: [1.0, 0.75, 0.75, 0.25, 0.5, 0.25]
7: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]

A_{i+1}	A_i
0	0
1	1

C_{i+1}	B_i
0	0
1	1

D_{i+1}	F_i
0	0
1	1

E_{i+1}	C_i	D_i
0	0	0
0	0	1
1	1	0
1	1	1

F_{i+1}	D_i	E_i
0	0	0
1	0	1
0	1	0
0	1	1

B_{i+1}	A_i	C_i	D_i
0	0	0	0
1	1	0	0
1	0	1	0
0	0	0	1
1	1	1	0
0	1	0	1
0	0	1	1
0	1	1	1

Table 1: The propagation matrix for the given graph.

```
39: [1.0, 0.75, 0.75, 0.25, 0.5, 0.25]
13: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
18: [0.0, 0.5, 0.5, 0.25, 0.5, 0.25]
19: [1.0, 0.75, 0.75, 0.25, 0.5, 0.25]
23: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
36: [0.0, 0.5, 0.5, 0.25, 0.5, 0.25]
26: [0.0, 0.5, 0.5, 0.25, 0.5, 0.25]
63: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
55: [1.0, 0.7142857142857143, 0.7142857142857143, 0.2857142857142857, 0.42857142857142855, 0.2857142857142857]
31: [1.0, 0.75, 0.75, 0.25, 0.5, 0.25])
```

Code:

Listing 1: Source code of the script `BooleanNode.py`

```python
# Node class, assignment 1
class BooleanNode:
    def __init__(self, identifier):
        """
        Sets node id and initialize empty node list that references its connected nodes
        """
        self.id = identifier
        self.nodelist = []
        self.state = 0
```

Listing 2: Source code of the script `BooleanNetwork.py`

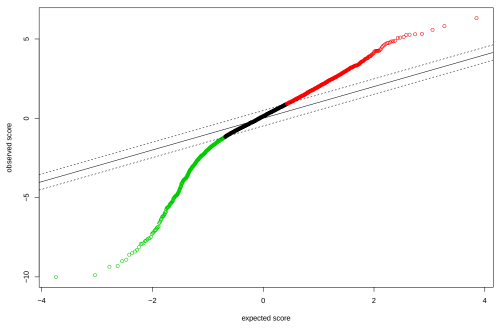
```python
# Node class, assignment 6
from BooleanNode import BooleanNode
import itertools
from collections import defaultdict

class BooleanNetwork:
    def __init__(self):
        self.nodelist = {}
        self.state = 0
        self.attractos_basins = defaultdict(list)
        self.attractos_length = {}
        self.attractor_distribution = defaultdict(list)
        self.attractor_coverage = {}

        #node A
        node = BooleanNode(0)
        node.nodelist = [1, 1, 0, 0, 0, 0]
        self.nodelist[0] = node
        # node B
        node = BooleanNode(1)
        node.nodelist = [0, 0, 1, 0, 0, 0]
        self.nodelist[1] = node
        # node C
        node = BooleanNode(2)
        node.nodelist = [0, 1, 0, 0, 1, 0]
        self.nodelist[2] = node
        # node D
        node = BooleanNode(3)
        node.nodelist = [0, -3, 0, 0, -3, -3]
        self.nodelist[3] = node
        # node E
        node = BooleanNode(4)
        node.nodelist = [0, 0, 0, 0, 0, 1]
        self.nodelist[4] = node
        #node F
        node = BooleanNode(5)
        node.nodelist = [0, 0, 0, 1, 0, 0]
        self.nodelist[5] = node

    def propagation(self, initstart):

        #initializing
        loop_states = []
        self.state = initstart

        #propagate until a loop is found
        while self.state not in loop_states:
            loop_states.append(self.state)
            seq_state = BooleanNetwork.getSequenceFromInt(self, self.state)
            self.state = 0
            post_state = 6 * [0]

            for n in range(len(seq_state)):
                if seq_state[n] == 1:
                    post_state = [i + j for i, j in zip(post_state, self.nodelist[n].nodelist)]

            for i in range(len(post_state)):
                if post_state[i] > 0:
                    self.state += 2**i

            if self.state == 0:
                break


        if self.state != 0:
            self.attractos_basins[self.state].append(initstart)
            self.attractos_length[self.state] = len(loop_states) - loop_states.index(self.state)

        loop_states.append(self.state)
        return loop_states

    def getSequenceFromInt(self, i):
            """Function to print binary number
            for the input decimal using recursion"""
            sequence = []
            while i > 0:
                sequence.append(i % 2)
                i = i // 2

            for i in range(len(sequence),6):
                sequence.append(0)
            return sequence

    def compute_prop_for_all(self):
        self.attractos_basins = defaultdict(list)
        self.attractos_length = {}
        all_possible_intial_states = []
        for i in range(1, 7):
            all_possible_intial_states.append([list(t) for t in list(itertools.combinations([0, 1, 
        for i in range(len(all_possible_intial_states)):
            for j in range(len(all_possible_intial_states[i])):
                init_state = 0
                for k in all_possible_intial_states[i][j]:
                    init_state += 2**k
                prop = BooleanNetwork.propagation(self, init_state)
                if prop[len(prop) - 1] > 0:
                    BooleanNetwork.compute_distr_nodes(self,prop, prop.index(prop[len(prop)-1]))
                    self.attractor_coverage[prop[len(prop) - 1]] = len(self.attractos_basins[prop[le

    def compute_distr_nodes(self, prop, start):
        num_nodes = [0] * 6
        for l in range(start, len(prop) - 1):
            seq = BooleanNetwork.getSequenceFromInt(self, prop[l])
            num_nodes = [x + y for x, y in zip(seq, num_nodes)]
            orbit_length = float(self.attractos_length[prop[len(prop)-1]])
        self.attractor_distribution[prop[len(prop) - 1]] = [x/orbit_length for x in num_nodes]



if __name__ == "__main__":
    network = BooleanNetwork()
    exercise = [1,4,21,33]
    for i in exercise:
        liststates = BooleanNetwork.propagation(network, i)
        print(liststates)
    BooleanNetwork.compute_prop_for_all(network)
    print(network.attractos_length)
    print(network.attractos_basins)
    print(network.attractor_coverage)
    print(network.attractor_distribution)
```

Exercise 6.2: Differential Expression Analysis

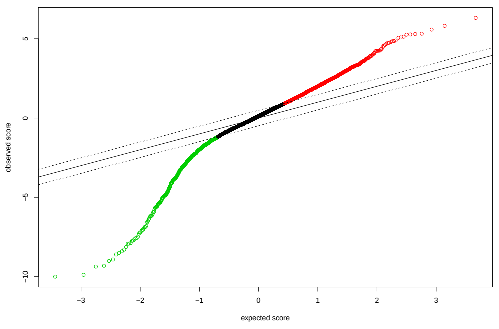
To run the differential expression analysis, the code shown in listing 3 has been used. `SAM` was then applied with 4 different settings, first switching the FDR ration between 5 and 20%, secondly by changing the number of permutations from 100 to 1000. Changing the FDR did not influence the top 10 up and down regulated genes, which are shown in Table 2 and Table 3. But, as expected, it changed the number of up and down regulated genes from 1482 to 676 (FDR level: 5% and 20%) for the up regulated and from 926 to 519 for the down ones.
The number of permutations also did not influenced the top 10 deregulated genes, but the overall distribution of deregulation scores (shown in Figure 1).

Listing 3: Source code of the script `BI3_EX_6_2`

```
# needed to install impute using bioclite
# source("https://bioconductor.org/biocLite.R")
# biocLite("impute")
# biocLite("preprocessCore")
if(!require(data.table)){
  install.packages("data.table")
}
if(!require(samr)){
  install.packages("samr")
}
if(!require(preprocessCore)){
  install.packages("preprocessCore")
}

# read data
dat = fread("ms_data.txt")

#### log2-transfrom ####
dat[,1:9] = log2(dat[,1:9])

#### quantile normalization ####
mat = normalize.quantiles(as.matrix(dat[,1:9]))


#### SAM ####

y<-c(rep(1,3),rep(2,6))    #define classes, 3 controles, 6 rnas
####  FDR ####
# frd 5%
samfit5 = SAM(mat,y,resp.type ="Two_class_unpaired", fdr.output = 0.05, nperms = 1000, genenames = d
plot(samfit5)

#down and up regulated genes
samfit5$siggenes.table$genes.up[1:10,]
samfit5$siggenes.table$genes.lo[1:10,]


# frd 20%
samfit20 = SAM(mat,y,resp.type ="Two_class_unpaired", fdr.output = 0.20, nperms = 1000, genenames =
plot(samfit20)

#down and up regulated genes
samfit20$siggenes.table$genes.up[1:10,]
samfit20$siggenes.table$genes.lo[1:10,]

samfit20$siggenes.table$ngenes.up
samfit5$siggenes.table$ngenes.up

samfit20$siggenes.table$ngenes.lo
samfit5$siggenes.table$ngenes.lo

# -> fdr does not change anything in top 10 up and down
#    changes the number of dregulated genes drastically

#### nperms ####
# nperms 1000
samfit1000 = SAM(mat,y,resp.type ="Two_class_unpaired", fdr.output = 0.20, nperms = 1000, genenames
plot(samfit1000)

#down and up regulated genes
samfit1000$siggenes.table$genes.up[1:10,]
samfit1000$siggenes.table$genes.lo[1:10,]

# nperms 100
samfit100 = SAM(mat,y,resp.type ="Two_class_unpaired", fdr.output = 0.20, nperms = 100, genenames =
plot(samfit100)

#down and up regulated genes
samfit100$siggenes.table$genes.up[1:10,]
samfit100$siggenes.table$genes.lo[1:10,]

# -> nperms does not influence the top 10 up and down,
```



(a) 100 iterations



(b) 1000 iterations

Figure 1: Plots of the overall regulation values for two different numbers of iterations in the SAM differential expression analysis. Only small changes can be spotted between the two plots, thus the number of iterations does not influence the analysis in a strong way.

Gene Name	Gene ID	Score(d)	Numerator(r)	Denominator(s+s0)	Fold Change	q-value(%)
IFRD1	477	6.314	1.416	0.224	1.052	0
PEG10	3916	5.812	1.337	0.23	1.045	0
DHRS7	1428	5.575	1.147	0.206	1.039	0
SULT1A4	2673	5.318	1.324	0.249	1.048	0
RALGAPA1	542	5.296	1.085	0.205	1.041	0
SH3RF1	3896	5.269	1.894	0.36	1.077	0
FADS3	1026	5.256	2.482	0.472	1.097	0
CHKA	2469	5.128	2.306	0.45	1.101	0
FAM129A	4822	5.083	1.587	0.312	1.058	0
COBL	1738	5.059	0.777	0.154	1.026	0

Table 2: Top 10 up regulated genes

```
#     but the overall distribution of the extreme values -> plots
```

Gene Name	Gene ID	Score(d)	Numerator(r)	Denominator(s+s0)	Fold Change	q-value(%)
ERGIC2;RLN3	4563	-10.002	-2.377	0.238	0.916	0
ITGAV	868	-9.888	-3.1	0.313	0.903	0
NT5E	2277	-9.367	-1.689	0.18	0.944	0
PLD3	254”	-9.314	-2.342	0.251	0.919	0
GPX8	4218	-9.013	-1.692	0.188	0.945	0
B4GALT1	1113	-8.92	-1.914	0.215	0.939	0
PCDH7	1051	-8.604	-3.652	0.424	0.858	0
CTSA	2101	-8.504	-1.779	0.209	0.941	0
CTSC	1215	-8.401	-1.704	0.203	0.945	0
MAN1B1	1415	-8.297	-1.67	0.201	0.944	0

Table 3: Top 10 down regulated genes