Bioinformatics III

Sixth Assignment

Thibault Schowing (2571837)
Wiebke Schmitt (2543675)

June 3, 2018

Exercise 6.1: Boolean Networks

All the listings are at the end of the exercise.

(a) **Weighted Interactions**

Table 1: Propagation Matrix

	F	E	D	C	B	A
F	0	0	1	0	0	0
E	1	0	0	0	0	0
D	-3	-3	0	0	-3	0
C	0	1	0	0	1	0
B	0	0	0	1	0	0
A	0	0	0	0	1	1

(b) **Implementation**

When does it make sense to stop the propagation and why?

We stop the propagation once we meet an already visited state. If we continue, we will just loop over and over again.

Which sequences do you get when you start from states 1, 4, 21, and 33?

Sequence with starting state 1: [1, 3, 7, 23, 55, 63, 13, 1]
Sequence with starting state 4: [4, 18, 36, 26, 4]
Sequence with starting state 21: [21, 51, 47, 13, 1, 3, 7, 23, 55, 63, 13]
Sequence with starting state 33: [33, 11, 5, 19, 39, 31, 5]

(c) **Periodic Orbits**
(1) List these orbits with their respective lengths and basins of attraction

To make things clearer let's recall the different definition. "If the attractor has only a single state it is called a point attractor, and if the attractor consists of more than one state it is called a cycle attractor. The set of states that lead to an attractor is called the basin of the attractor. States which occur only at the beginning of trajectories (no trajectories lead to them), are called garden-of-Eden states" [1]

[1] https://en.wikipedia.org/wiki/Boolean_network#Attractors

Table 2: List of states, orbith length, Cycle attractor and relative coverage of the basin of attraction. The basin of attraction's coverage includes the steps before the cycle attractor.

Start State	**Period**	**Basin**	**Attractor**	**Basin Coverage**
0	1	[]	[0]	1.5625%
1	7	[]	[1, 3, 7, 23, 55, 63, 13]	10.9375%
2	4	[2]	[4, 18, 36, 26]	7.8125%
3	7	[]	[3, 7, 23, 55, 63, 13, 1]	10.9375%
4	4	[]	[4, 18, 36, 26]	6.25%
5	4	[]	[5, 19, 39, 31]	6.25%
6	1	[6, 22, 54, 62, 12]	[0]	9.375%
7	7	[]	[7, 23, 55, 63, 13, 1, 3]	10.9375%
8	1	[8]	[0]	3.125%
9	7	[9]	[1, 3, 7, 23, 55, 63, 13]	12.5%
10	4	[10]	[4, 18, 36, 26]	7.8125%
11	4	[11]	[5, 19, 39, 31]	7.8125%
12	1	[12]	[0]	3.125%
13	7	[]	[13, 1, 3, 7, 23, 55, 63]	10.9375%
14	4	[14]	[4, 18, 36, 26]	7.8125%
15	4	[15]	[5, 19, 39, 31]	7.8125%
16	1	[16, 32, 8]	[0]	6.25%
17	4	[17, 35, 15]	[5, 19, 39, 31]	10.9375%
18	4	[]	[18, 36, 26, 4]	6.25%
19	4	[]	[19, 39, 31, 5]	6.25%
20	1	[20, 50, 44, 8]	[0]	7.8125%
21	7	[21, 51, 47]	[13, 1, 3, 7, 23, 55, 63]	15.625%
22	1	[22, 54, 62, 12]	[0]	7.8125%
23	7	[]	[23, 55, 63, 13, 1, 3, 7]	10.9375%
24	1	[24]	[0]	3.125%
25	7	[25]	[1, 3, 7, 23, 55, 63, 13]	12.5%
26	4	[]	[26, 4, 18, 36]	6.25%
27	4	[27]	[5, 19, 39, 31]	7.8125%
28	1	[28]	[0]	3.125%
29	7	[29]	[1, 3, 7, 23, 55, 63, 13]	12.5%
30	4	[30]	[4, 18, 36, 26]	7.8125%
31	4	[]	[31, 5, 19, 39]	6.25%
32	1	[32, 8]	[0]	4.6875%
33	4	[33, 11]	[5, 19, 39, 31]	9.375%
34	1	[34, 12]	[0]	4.6875%
35	4	[35, 15]	[5, 19, 39, 31]	9.375%
36	4	[]	[36, 26, 4, 18]	6.25%
37	4	[37, 27]	[5, 19, 39, 31]	9.375%
38	4	[38, 30]	[4, 18, 36, 26]	9.375%
39	4	[]	[39, 31, 5, 19]	6.25%
40	1	[40, 8]	[0]	4.6875%

Start State	Period	Basin	Attractor	Basin Coverage
41	7	[41, 9]	[1, 3, 7, 23, 55, 63, 13]	14.0625%
42	1	[42, 12]	[0]	4.6875%
43	7	[43]	[13, 1, 3, 7, 23, 55, 63]	12.5%
44	1	[44, 8]	[0]	4.6875%
45	7	[45, 9]	[1, 3, 7, 23, 55, 63, 13]	14.0625%
46	1	[46, 12]	[0]	4.6875%
47	7	[47]	[13, 1, 3, 7, 23, 55, 63]	12.5%
48	1	[48, 40, 8]	[0]	6.25%
49	7	[49, 43]	[13, 1, 3, 7, 23, 55, 63]	14.0625%
50	1	[50, 44, 8]	[0]	6.25%
51	7	[51, 47]	[13, 1, 3, 7, 23, 55, 63]	14.0625%
52	1	[52, 58, 12]	[0]	6.25%
53	7	[53, 59]	[13, 1, 3, 7, 23, 55, 63]	14.0625%
54	1	[54, 62, 12]	[0]	6.25%
55	7	[]	[55, 63, 13, 1, 3, 7, 23]	10.9375%
56	1	[56, 8]	[0]	4.6875%
57	7	[57, 9]	[1, 3, 7, 23, 55, 63, 13]	14.0625%
58	1	[58, 12]	[0]	4.6875%
59	7	[59]	[13, 1, 3, 7, 23, 55, 63]	12.5%
60	1	[60, 8]	[0]	4.6875%
61	7	[61, 9]	[1, 3, 7, 23, 55, 63, 13]	14.0625%
62	1	[62, 12]	[0]	4.6875%
63	7	[]	[63, 13, 1, 3, 7, 23, 55]	10.9375%

(2) Give the relative coverages of the state space by the basins of attraction.

The coverages for each separate basins of attraction + cycle attractor are given in the table 2. In table 3 we give the coverage of each cycle attractor.

Table 3: Relative coverage of the cycle attractors. Details of the basins in table 4.

[0] :	23	35.9375 %
[1, 3, 7, 23, 55, 63, 13] :	21	32.8125 %
[4, 18, 36, 26] :	9	14.0625 %
[19, 39, 31, 5] :	11	17.1875 %

Table 4: State space occupation of the basins of attraction. Details of the basins leading to attractor. Here we included the attractor in the basin.

[0] :	[0, 6, 8, 12, 16, 20, 22, 24, 28, 32, 34, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62]	35.9375 %
[1, 3, 7, 23, 55, 63, 13] :	[[1, 3, 7, 9, 13, 21, 23, 25, 29, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63]]	32.8125 %
[4, 18, 36, 26] :	[2, 4, 36, 38, 10, 14, 18, 26, 30]	14.0625 %
[19, 39, 31, 5] :	[33, 35, 5, 37, 39, 11, 15, 17, 19, 27, 31]	17.1875 %

(d) **Interpretation**

(1) Give the attractors in terms of active genes and characterize them with a few words

In the listing 1 the binary transitions are presented for each attractors.

Attractor [0] : No gene are activated in this attractor and there is only one period. We can see that the states leading to this attractor are the one when **D** is activated and shuts down genes **B**, **E** and **F** even if **C** is activated, in those case it does not get activated again by **B** as in attractor [4, 18, 36, 26]. Here **A** is not activated and there is no cycle between **B** and **C**.

Attractor [1, 3, 7, 23, 55, 63, 13] : Here gene **A** is activated and keeps activating **B**. Thus, whenever **D** is activated and shuts down genes **B**, **E** and **F**, **B** is reactivated again by **A**.

Attractor [4, 18, 36, 26] : This cycle happens when gene **B** and **C** are not active at the same time and keep activating each other one step after another.

Attractor [5, 19, 39, 31] : This attractor is the opposite of attractor [1, 3, 7, 23, 55, 63, 13] for gene **B**, **C** and **D**. Gene **A** is also always activated but the fact that **C** is activated in the first place, shifts the activation of **D** earlier and avoids the total activation before **D** inhibits genes **B**, **E** and **F**.

Listing 1: Output - Binary evolution in the orbits and percentages

```
Current orbit:  [0]
Binary evolution:
[0, 0, 0, 0, 0, 0]
Average occupancy:
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Current orbit:  [1, 3, 7, 23, 55, 63, 13]
Binary evolution:
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 1, 1]
[0, 1, 0, 1, 1, 1]
[1, 1, 0, 1, 1, 1]
[1, 1, 1, 1, 1, 1]
[0, 0, 1, 1, 0, 1]
Average occupancy:
 [0.2857142857142857, 0.42857142857142855, 0.2857142857142857,
     0.7142857142857143, 0.7142857142857143, 1.0]

Current orbit:  [4, 18, 36, 26]
Binary evolution:
[0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 1, 0]
[1, 0, 0, 1, 0, 0]
[0, 1, 1, 0, 1, 0]
Average occupancy:
 [0.25, 0.5, 0.25, 0.5, 0.5, 0.0]

Current orbit:  [5, 19, 39, 31]
Binary evolution:
[0, 0, 0, 1, 0, 1]
[0, 1, 0, 0, 1, 1]
[1, 0, 0, 1, 1, 1]
[0, 1, 1, 1, 1, 1]
Average occupancy:
 [0.25, 0.5, 0.25, 0.75, 0.75, 1.0]
```

(2) Which are the special genes and what are their respective effects on the behavior of the network? For this, explain what is determining the period of the orbits. Further, compare the two shorter orbits which each other. Which gene is responsible for the difference?

Without genes **A** and **C**, the network shuts down. **A** is particular as it is not inhibited by **D** like the others and because it activates itself. As **A** cannot be activated by any other gene, it leads to or odd states, where **A** is active or even states where it is not. It cannot be activated in the middle of a sequence.

D is particular because it deactivates **B**, **E** and **F** what has a big effect on the network. **B** and **C** can propagate the activation to the network but they have to be synchronized correctly. If for instance **C** and **D** are activated, the network will be shut down.

Listing 2: boolean_network.py

```
import copy
import collections

class BooleanNetwork:
    def __init__(self, matrix):
        # Propagation Matrix
        self.matrix = matrix

        # Number of genes/nodes
        self.nb_bits = len(matrix)

        # Number of possible states (in our example 2**6 = 64)
        self.nb_states = int(2**self.nb_bits)

        # For each gene, we can set a threshold. Here we assume it's 0 for all of
            them
        self.threshold = [0] * self.nb_bits


    def int_to_binary(self, n):
        """
        Convert an integer to binary
        /!\ Uses the variable: SELF.NB_BITS !

        :Source: https://stackoverflow.com/questions/10411085/converting-integer-to
            -binary-in-python
        :param n: Integer
        :return: map object -> use list comprehension to print/loop in etc
        """
        ret = bin(n)[2:].zfill(self.nb_bits)
        return list(map(int, format(ret)))

    def binary_to_int(self, n):
        """
        Converts binary number (list) to Integer
        :param n: List of bits
        :return: the list converted to int
        """
        ret = ""
        for bit in n:
            ret += str(bit)
        ret_int = int(ret,2)
        return ret_int


    def get_states_sequence(self, start_state):
        """
        From a passed starting state (INT), calculates all the following states and
             stops when a loop is detected.
        :param start_state: INT - number of the starting state
        :return: The sequence of states,
        """

        binary_current_state = self.int_to_binary(start_state)

        binary_next_state = [0] * self.nb_bits
        states_sequence = [start_state]

        state_not_visited = True

        # While the state haven't already been visited...(while not orbit)
        while state_not_visited:

            for i in range(0, self.nb_bits):
                sum_next_state = 0
                for j in range(0, self.nb_bits):
```

```
                    # print("Fucking i: ", i, " and j: ", j)
                    # print(binary_current_state[j] * self.matrix[j][i])
                    sum_next_state += binary_current_state[j] * self.matrix[j][i]

                # Now we can set the next state for this gene
                if sum_next_state > self.threshold[i]:
                    binary_next_state[i] = 1
                else:
                    binary_next_state[i] = 0

            # If the next state doesn't already exist, we add it to the states list and continue
            # If not, we add it and end the while loop

            #print("CURRENT STATE: ", binary_next_state)
            if self.binary_to_int(binary_next_state) not in states_sequence:
                states_sequence.append(self.binary_to_int(binary_next_state))
            else:
                states_sequence.append(self.binary_to_int(binary_next_state))
                state_not_visited = False

            # The state now (starting state) is the next state

            binary_current_state = copy.copy(binary_next_state)

        return states_sequence

    def orbit(self, start_state):
        """
        Return many things, orbit is a random name here
        :param start_state:
        :return: see comments below
        """
        # Sequence from start state -> stabilization
        sequence = self.get_states_sequence(start_state)

        # At which state it closes its orbit
        closure = sequence[-1]

        # Basin = states leading to periodic orbits (cyclic attractor)
        partial_basin = sequence[0:sequence.index(closure)]

        # Periodic orbit (cyclic attractor) of the sequence
        periodic_orbit = sequence[sequence.index(closure):len(sequence) - 1]

        # Size of the orbit
        orbit_size = len(sequence) - sequence.index(closure) - 1

        return orbit_size, periodic_orbit, closure, partial_basin


    def count_attractors(self):
        """
        # Returns a counter containing the sets of attractors and their occurence
        # In our case:
        # Counter({frozenset({0}): 23,
        #          frozenset({1, 3, 7, 13, 55, 23, 63}): 21,
        #          frozenset({39, 19, 5, 31}): 11,
        #          frozenset({18, 26, 4, 36}): 9})
        #
        # Notice that unfortunately here the order of the attractor is not maintained
        # Source: https://stackoverflow.com/questions/37295981/python-creating-a-dictionary-with-key-as-a-set-and-value-as-its-count
        #

        :return:
```

```
        """
        all_attractors_list = [sorted(self.orbit(i)[1]) for i in range(self.
            nb_states)]
        attractors_count = collections.Counter(frozenset(x) for x in
            all_attractors_list)

        return attractors_count


    def average_occupancies_in_orbit(self, orbit, print_steps = None):
        """
        Count occurence of ABCDEF activated in each states
        :param orbit:
        :return: list of percentages like [0.25, 0.5, 0.25, 0.5, 0.5, 0.0]
        """
        total_bits = [0 for _ in range(0, self.nb_bits)]
        percentages = [0.0 for _ in range(0, self.nb_bits)]

        # For every state, convert to binary and increment the occupancy
        for state in orbit:
            bin = self.int_to_binary(state)

            # if the optional parameter is true, print the binary states
            # allow to see more clearely the evolution of the different gene
                activation
            if print_steps:
                print(bin)
            #Increment here
            for i in range(0, len(bin)):
                total_bits[i] += bin[i]
        # Calculate percentages
        total_elem = len(orbit)
        for i in range(0, self.nb_bits):
            percentages[i] = total_bits[i] / total_elem

        return percentages


    def get_basin_of_attraction(self):
        """

        :return:
        """

        sequence_set = collections.defaultdict(list)

        for i in range(0, self.nb_states):
            orbit = self.orbit(i)
            unique_orbit_string = str(sorted(orbit[1]))
            # We have the "previous steps" and the cyclic attractor -> add the
                previous steps to the set corresponding to the attractor to have
                the full basin
            sequence_set[unique_orbit_string].extend(orbit[3])
            # To have the whole basin, we add also the cycle
            sequence_set[unique_orbit_string].extend(orbit[1])

        for e in sequence_set:
            sequence_set[e] = list(set(sequence_set[e]))

        return sequence_set
```

Exercise 6.2: Differential Expression Analysis

(a) **A**

Listing 3: r

```
print("Assignment 6 - Schmitt Schowing")

# Install the packages

#source("http://bioconductor.org/biocLite.R")
#biocLite("impute")
#biocLite("samr")
#biocLite("preprocessCore")

library(preprocessCore)
library(samr)

# Reade the data
mydata = read.csv('./ms_data.txt', sep = '\t')
df <- as.data.frame(mydata, col.names = mydata[1,], cut.names = FALSE)

#dim(df)
#names(df)
#str(df)

# Log-transformation of the data part
df.transformed <- df
df.transformed[, 1:9] <- log(df[1:9], 2)

# Normalization - extract matrix and replace
df.quantile_normalized <-df.transformed
x = data.matrix(df.transformed[,1:9])
x <- normalize.quantiles(x)


df.quantile_normalized[,1:9] <- x

# Differential expression analysis
x <- subset(df.quantile_normalized, select=c("control.1","control.2", "control
    .3", "rna1.1", "rna1.2", "rna1.3"))
x2 <- subset(df.quantile_normalized, select=c("control.1","control.2", "
    control.3", "rna2.1", "rna2.2", "rna2.3"))

x <- as.matrix(x)
x2 <- as.matrix(x2)

y <- subset(df["Gene.names"])


dim(x)


# First experiment (first set of rna)
df.analysis1 <- SAM(x,y = c(1,1,1,2,2,2) ,
                    resp.type=c("Two class unpaired"),
                    genenames = df[["Gene.names"]],
                    s0=NULL,
                    s0.perc=NULL,
                    nperms=100,
                    center.arrays=FALSE,
                    regression.method=c("standard","ranks"),
                    knn.neighbors=10,
                    random.seed=NULL,
                    logged2 = TRUE,
                    fdr.output = 0.20,
```

```
                    eigengene.number = 1)

summary(df.analysis1)
df.analysis1$siggenes.table

# Open file for output
fileConn<-file("outputSAM.txt", "w")

write(c("Assignment 6 - SAM function output with variating fdr.output and
    nperms"), fileConn, append = TRUE)

# Varies the fdr.output parameter from 0.1 to 1 with a 0.1 increment
# varies the nperms parameter from 100 to 1000 witn a 100 increment

for(i in 1:10){
  for(j in 1:10){
    fdr = 0.1 * i
    per = 100 * j

    line = "\n\n"
    write(line, fileConn, append = TRUE)

    line = paste(c("fdr.output", fdr), collapse = " ")
    write(line, fileConn, append = TRUE)

    line = paste(c("nperms", per), collapse = " ")
    write(line, fileConn, append = TRUE)



    df.analysis1 <- SAM(x2,y = c(1,1,1,2,2,2) ,
                        resp.type=c("Two class unpaired"),
                        genenames = NULL,
                        s0=NULL,
                        s0.perc=NULL,
                        nperms=per,
                        center.arrays=FALSE,
                        regression.method=c("standard","ranks"),
                        knn.neighbors=10,
                        random.seed=NULL,
                        logged2 = TRUE,
                        fdr.output = fdr,
                        eigengene.number = 1)

    line = "\nUp-Regulated protein\n"
    write(line, fileConn, append = TRUE)
    write.table(df.analysis1$siggenes.table$genes.up, file = fileConn, append
        = TRUE, quote = TRUE, sep = " ",
                col.names = TRUE, qmethod = c("escape", "double"),
                fileEncoding = "")
    line = "\nDown-Regulated protein\n"
    write(line, fileConn, append = TRUE)
    write.table(df.analysis1$siggenes.table$genes.lo, file = fileConn, append
        = TRUE, quote = TRUE, sep = " ",
                col.names = TRUE, qmethod = c("escape", "double"),
                fileEncoding = "")
  }

}

# Close file
close(fileConn)

df.analysis1$siggenes.table$genes.up
```

```
# From the tables, we get the following row
df[3004,]
df[2083,]
df[477,]
df[3861,]
df[1790,]
df[2376,]
df[898,]
df[3332,]
df[1428,]
df[5246,]

# And the downs
df[2018,]
df[3162,]
df[3325,]
df[1669,]
df[3026,]
df[868,]
df[4009,]
df[3264,]
df[309,]
df[1051,]
```

(b) **Top ten up and down-regulated proteins**
In this scrip, we vary the parameters *fdr.output* and *nperms*. The *fdr.output* varies between 0.1 and 1 (0.1 increment) and the permutations between 100 and 1000 (100 increment). The up-regulated and down-regulated proteins are stored in a file called *outputSAM.txt*.
During these variations, the top 10 proteins (up and down) do not vary but the fold-change does. Here is a example of the output with the parameters *fdr.output = 0.4* and *nperms = 400*. All the informations are saved in the file *outputSAM.txt* for each steps with the code in listing 3.

Table 5: Top ten Up regulated proteins

Gene ID	Protein	Gene Name	Fold Change
3004	Disabled Homolog 2	DAB2	3.074
2083	Heme oxygenase 1	HMOX1	3.776
477	Interferon-related developmental regulator 1	IFRD1	3.003
3861	EPM2A-interacting protein 1	EPM1AIP1	8.613
1790	Disabled homolog 1	DAB1	3.206
2376	Cellular retinoic acid-binding protein 2	CRABP2	2.263
898	Asparagine symthetase	ASNS	2.345
3332	EKC/KEOPS complex subunit LAGE3	LAGE3	3.011
1428	Dehydrogenase/reductase SDR family member 7	DHRS7	2.286
5246	Actin-related protein 10	ACTR10	2.649

Table 6: Top ten Down Regulated Protein

Gene ID	Protein	Gene Name	Fold Change
2018	Alpha-galactosidase A	GLA	0.245
3162	Receptor-type tyrosine-protein phosphatase eta	PTPRJ	0.303
3325	Protein disulfide-isomerase A5	PDIA5	0.148
1669	EGF-like repeat and discoidin I-like domain-containing protein 3	EDIL3	0.207
3026	Spectrin beta chain non-erythrocytic 1	SPTBN1	0.258
868	Integrin alpha-V heavy+light chains	IDGAV	0.142
4009	DnaJ homolog subfamily C member10	DNAJC10	0.186
3264	Spectrin alpha chain, non-erythrocytic1	SPTAN1	0.364
309	Cathepsin B light + heavy chains	CTSB	0.282
1051	Protocadherin-7	PCDH7	0.072