Hands-on tutorial on Boolean networks – Introduction

C. Müssel, L. Lausser, H. A. Kestler

Modeling Boolean networks

In *BoolNet*, models are described in a CSV format having the target genes in the first column and the logical transition functions in the second column:

TARGETS siehe exercises

```
targets, factors
<target gene 1>, <Boolean function>
<target gene 2>, <Boolean function>
```

Boolean networks can be assembled from natural-language statements on regulatory interactions. The following gives some examples of common statements:

Statement	*Boolean function*	*Notes*
Gene A is an input of the network	`A, A`	Modeled by a self-regulation
Gene C can be activated independently by gene A and gene B	`C, A | B`	Logical OR
Gene A and gene B jointly upregulate gene C	`C, A & B`	Logical AND
Gene A inhibits gene B	`B, !A`	Logical negation
Gene B is activated two time steps after gene A	`B, A[-2]`	Generalized time delays (`A` without an explicit delay is equivalent to `A[-1]`)
Gene B is activated by Gene A through factors C, D and E which are not included in the model.	`B, A[-4]`	Cascades can be subsumed by increasing the time delay for each member of the cascade.
Gene D is active if at least two of the genes A, B, C were active in the last time step	`D, sumgt(A,B,C,1)`	Other operators that can be used similarly include `all`, `any`, `maj`, `sumlt`, `sumis`
Gene B is active if gene A has been active in the majority of the previous three time steps	`B, maj[t=-3..-1](A[t])`	Similar to the above line, but based on a time range
Gene A is active for the first 5 time steps	`A, timelt(6)`	Temporal predicates also include `timegt`, `timeis`

Network simulation

In *BoolNet*, networks can be loaded via

```
> net <- loadNetwork("network.txt", symbolic=TRUE)
```

The regulatory dependencies in the network can be visualized using

```
> plotNetworkWiring(net)
```

To simulate the network using 1000 randomly generated initial states, use

```
> sim <- simulateSymbolicModel(net, startStates=1000)
```

The attractors of the network can be visualized by calling

```
> plotAttractors(sim)
```

The whole state transition graph can be plotted using

```
> plotStateGraph(sim)
```

To see the trajectory from a specific initial state (e.g. with genes A and B active) to the attractor, use

```
> # generate initial state -- all unspecified genes are set to 0
> state <- generateState(net, c("A" = 1, "B" = 1))
> plotSequence(network=net, startState=state)
```

In case of time delays of more than 1, it might be required to specify the history of states before the initial state. `generateState()` also supports the generation of sequences of start states:

```
> state <- generateState(net, list("A" = c(0,0,1), "B" = c(1,1,0)))
> plotSequence(network=net, startState=state)
```

Robustness analysis

To perform a robustness analysis by measuring the Hamming distances between the successor states of random initial states and their perturbed copies, type

```
> x <- perturbTrajectories(net,measure="hamming",numSamples=1000)
```

`x$value` holds the average normalized Hamming distance for the 1000 samples, while `x$stat` comprises the single distances for each of the samples.
The Hamming distance as a robustness measure can also be used for statistical testing. *BoolNet* includes a generic testing tool that compares properties of biological models to randomly generated models.
The following generates 1000 random Boolean networks with the same numbers of inputs for the genes as the biological network. It then calculates the average Hamming distance as above (specified by the test function `testTransitionRobustness()` that analyzes `numSamples=500` random state pairs) and determines in how many of these networks the average distance is less than the average distance in the true network:

```
> t <- testNetworkProperties(network=net,
+                            numRandomNets=1000,
+                            testFunction=testTransitionRobustness,
+                            testFunctionParams=list(numSamples=500),
+                            alternative="less")
```

It is also possible to determine the way random networks are generated. For example, the biologically plausible class of *canalyzing functions* can be generated by setting `functionGeneration=generateCanalyzing`.

More help

For all commands in *BoolNet*, a manual page is available (e.g. `?perturbTrajectories`).
A very detailed introduction to *BoolNet* can be accessed using

```
> vignette("BoolNet_package_vignette")
```