Comput Stat (2013) 28:19–36
DOI 10.1007/s00180-012-0324-2

ORIGINAL PAPER

Attractors in Boolean networks: a tutorial

Martin Hopfensitz · Christoph Müssel · Markus Maucher · Hans A. Kestler

Received: 21 September 2010 / Accepted: 22 March 2012 / Published online: 8 April 2012

Abstract Boolean networks are a popular class of models for the description of gene-regulatory networks. They model genes as simple binary variables, being either *expressed* or *not expressed*. Simulations of Boolean networks can give insights into the dynamics of cellular systems. In particular, stable states and cycles in the networks are thought to correspond to phenotypes. This paper presents approaches to identify attractors in synchronous, asynchronous and probabilistic Boolean networks and gives examples of their usage in the *BoolNet* R package.

Keywords Systems biology · Boolean networks · Attractors

1 Introduction

Gene expression in living systems is controlled by complex interactions of genes and gene products. Genes and their expression products influence the expression of other genes and, as a whole, form complex gene-regulatory networks. In order to gain insight into the functioning of biological organisms, researchers need intuitive and compact models of these networks.

Dynamic models of gene regulation are able to capture the behaviour of the networks over time. Modeling approaches include ordinary differential equations (e.g. Goodwin 1963; Wawra et al. 2007) and dynamic Bayesian models (e.g. Dojer et al.

Martin Hopfensitz and Christoph Müssel contributed equally.

M. Hopfensitz · C. Müssel · M. Maucher · H. A. Kestler (✉)
Research Group Bioinformatics and Systems Biology,
Institute of Neural Information Processing,
University of Ulm, 89069 Ulm, Germany
e-mail: hans.kestler@uni-ulm.de

2006; Grzegorczyk et al. 2010). Another popular class of dynamic models are Boolean networks (BN) (Kauffman 1969, 1993). This qualitative approach models genes as Boolean variables, i.e. a gene can be either transcribed (ON, 1) or not transcribed (OFF, 0). Although this clearly is a simplification, Boolean networks approximate the real nature of gene-regulatory networks well: The general building blocks that have been identified in genetic networks constitute different types of robust switching elements. Their simple dynamic behaviour can be imitated by Boolean models (Bornholdt 2005). Furthermore, concentration levels in gene-regulatory networks presumably behave according to a Hill function (de Jong 2002). For many values of the Hill coefficient, the function has a sigmoidal form and can be approximated by a dichotomous step function (Glass and Kauffman 1973).

There are three main types of Boolean networks:

Synchronous Boolean networks (Kauffman 1969, 1993) consist of a set of Boolean variables

$$X = \{X_1, \ldots, X_n\}$$

and a set of transition functions

$$F = \{f_1, \ldots, f_n\},$$

one for each variable. These transition functions map an input of the Boolean variables in X to a Boolean value (0 or 1). We call a Boolean vector $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ the *state* of the network at time t. Then, the successor state $\mathbf{x}(t+1)$ is calculated by applying the n-dimensional synchronous state transition function $\mathbf{T}_{sync}$ to $\mathbf{x}(t)$:

$$\mathbf{x}(t+1) = \mathbf{T}_{sync}(\mathbf{x}(t))$$

with $\mathbf{T}_{sync} = (T_1, \ldots, T_n)$ and $T_i(\mathbf{x}) = f_i(\mathbf{x})$. That means that *all* functions are applied simultaneously.

In a biological context, the Boolean variables X correspond to the genes, and the transition functions model the dependencies among the genes. In the synchronous model, the assumption is that all genes are updated at the same time.

Asynchronous Boolean networks (Thomas 1991; Harvey and Bossomaier 1997) also consist of a set of variables X and a set of transition functions F as above. Yet, at each point of time t, only *one* of the transition functions $f_{i^*} \in F$ is chosen at random, and the corresponding Boolean variable is updated. This corresponds to the assumption that in a genetic network, gene expression levels are likely to change at different points of time. Formally, the asynchronous state transition function $\mathbf{T}^{(i^*)}_{async} = \left(T_1^{(i^*)}, \ldots, T_n^{(i^*)}\right)$ associated with a transition of gene i^* is defined as follows:

$$T_i^{(i^*)}(\mathbf{x}) = \begin{cases} f_i(\mathbf{x}) & i = i^* \\ x_i & i \neq i^* \end{cases}$$

There are n different state transition functions, one for each gene i^*. The successor state is calculated by applying the chosen function to the current state, i.e.

$$\mathbf{x}(t+1) = \mathbf{T}_{async}^{(i^*)}(\mathbf{x}(t))$$

Probabilistic Boolean networks (Shmulevich et al. 2002) allow for specifying more than one transition function per variable/gene in X. Each of these functions has a probability of being chosen, where the probabilities of all functions for one variable sum up to 1. Formally

$$F = \left\{ \left\{ (f_{11}, p_{11}), \ldots, \left(f_{1k_1}, p_{1k_1}\right) \right\}, \ldots, \left\{ (f_{n1}, p_{n1}), \ldots, \left(f_{nk_n}, p_{nk_n}\right) \right\} \right\}$$

where k_i is the number of alternative transition functions for variable i, and p_{ij} is the probability that function j is chosen for variable i, with $\sum_{j=1}^{k_i} p_{ij} = 1$. A state transition is performed by choosing a network $N = (f_{ij_1^*}, \ldots, f_{ij_n^*})$ according to these probabilities and applying a synchronous update. The transition function $\mathbf{T}_{prob}^{(N)} = \left(T_1^{(N)}, \ldots, T_n^{(N)}\right)$ with respect to this chosen network N is defined as $T_i^{(N)}(\mathbf{x}) = f_{ij_i^*}(\mathbf{x})$. The successor state is calculated by applying this function to the current state:

$$\mathbf{x}(t+1) = \mathbf{T}_{prob}^{(N)}(\mathbf{x}(t))$$

Boolean networks can be constructed on the basis of qualitative literature statements (Zhou et al. 2009). In a typical scenario, an expert collects natural-language statements on gene dependencies and transfers them into Boolean rules (see e.g. Albert and Othmer 2003). For example, the statement "Gene 1 inhibits Gene 2" can be expressed as the Boolean rule $Gene_2(t+1) = \neg Gene_1(t)$. This yields one or several hypothetical networks whose dynamical behaviour can be compared to measurements from wet-lab experiments. By iteratively refining the models, one can finally construct a network that matches these expectations (Bornholdt 2008). Another way of inferring Boolean networks is reconstruction from time series of expression measurements. Here, algorithms infer dependencies between genes by analyzing changes in gene expression over time. Popular algorithms include *Best-Fit Extension* (Lähdesmäki et al. 2003) and *REVEAL* (Liang et al. 1998). Due to uncertainty in measurements and in the discretization process and due to the impossibility of obtaining time series that cover enough time points to describe one distinct network, reverse engineering usually generates many candidate networks. Thus, network modeling always involves manual inspection by human experts.

Simulations of Boolean networks can provide insight into the dynamics of the underlying gene-regulatory networks (Bornholdt 2005). In this context, the identification of recurrent cycles of states—so-called *attractors*—is of particular interest: As a network is trapped in these cycles as soon as one of its states is entered, they comprise

the states in which the network resides most of the time. It is assumed that attractors in models of gene-regulatory networks are linked to phenotypes (Kauffman 1993). For example, several studies involving Boolean networks have associated attractors with cell cycle phases (Fauré et al. 2006; Li et al. 2004; Orlando et al. 2008). Albert and Othmer associate an attractor in a Boolean model of Drosophila melanogaster with the wild-type gene-expression patterns of the segment polarity genes (Albert and Othmer 2003). Huang considers three states of adult tissue cells (growth, cell differentiation and apoptosis) and claims that cycles of multiple states correspond to growth of the cell, whereas single-state attractors correspond to differentiation or apoptosis (programmed cell death) (Huang 1999). As disruptions of the balance of these cell states are a characteristic of neoplasia (i.e. tumors), analyses of Boolean cell models could also provide new insights into the origins of cancer (Orlando et al. 2008; Sahin et al. 2009).

Transitions from all states in a Boolean network eventually lead to an attractor, as the number of states in a network is finite. All states that lead to a certain attractor form its *basin of attraction*. Biologically relevant attractors often have a large basin of attraction (Huang 1999; Li et al. 2004).

In synchronous and asynchronous networks, attractors can be identified directly by iterating the state transitions. The types of attractors differ depending on the network type:

Simple cycles are loops of states—that is, any state in the cycle is reached again after a fixed number of successive state transitions (the *cycle length*). Formally, $\mathbf{x}(t) = \mathbf{x}(t+l)$ for a cycle length l. In synchronous Boolean networks, such cycles are attractors, as there is exactly one possible successor state for each state. In asynchronous networks, there is usually more than one successor state, so that it is possible to leave the cycle. In the case where asynchronous updates yield only a single successor state—which means that only one gene changes its value in a synchronous state transition from $\mathbf{x}(t)$ to $\mathbf{x}(t+1)$—a simple cycle is also an asynchronous attractor (i.e. a special complex attractor). In all other cases, simple cycles do not form attractors in asynchronous networks. A simple cycle is depicted in Panel A of Fig. 1.

Complex or loose attractors are formed by overlapping loops which occur when there is possibly more than one successor state due to asynchronous updates. Precisely, a complex attractor is a set of states in which all asynchronous state transitions lead to another state in the set, and each state in the set can be reached from all other states in the set. For a formal definition, we denote the set $FS(\mathbf{x}) = \left\{\mathbf{y} \mid \exists k \geq 1, \exists i_1^*, \ldots, i_k^* : \mathbf{T}_{async}^{(i_1^*)} \circ \cdots \circ \mathbf{T}_{async}^{(i_k^*)}(\mathbf{x}) = \mathbf{y}\right\} \cup \{\mathbf{x}\}$ as the *Forward reachable set* of state $\mathbf{x}$. A complex attractor is a set of states S with $S = FS(\mathbf{s})$ for all $\mathbf{s} \in S$. Panel B of Fig. 1 shows an example of a complex attractor. A simple loop in a synchronous network may be part of a complex loop in an asynchronous model, but this is not necessarily the case (Garg et al. 2008b).

Steady-state attractors are attractors that consist of only one state. All transitions from this state result in the state itself. These attractors are the same both for synchronous and asynchronous update of a network. Steady states are a special case of both simple cycles and complex attractors. A steady state is shown in Panel C of Fig. 1.

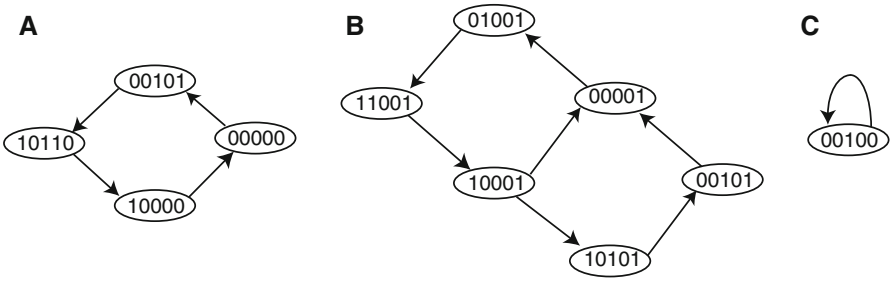

Fig. 1 **a** A simple cycle in a synchronous Boolean network. **b** A complex attractor in an asynchronous Boolean network. **C** A steady-state attractor in a synchronous or asynchronous Boolean network

In probabilistic networks, attractors are usually identified indirectly by performing Markov chain simulations (Shmulevich et al. 2002). This yields a list of attractor states and their probabilities, but does not give direct information on the number and structure of the attractors.

The dynamic properties of Boolean networks, in particular the number and size of attractors, have been studied extensively in the context of randomly generated synchronous Boolean networks. Originally proposed by Kauffman (1993), such networks are generated according to three parameters, n, k, and p. Here, n, is the total number of genes in the network, and k is the number of inputs for the transition functions of the genes. In Kauffman's model, k is the same for all genes. p defines the bias of the transition functions towards returning a 0 or a 1, i.e. it denotes the probability that a transition function returns 1 given a uniformly distributed input. These random Boolean networks (RBN) can be partitioned into three classes according to Kauffman (1993), Socolar and Kauffman (2003): In the *ordered regime*, almost all genes are frozen, and attractor cycles are short. Perturbations of the initial states of gene values in such networks die out. In the *chaotic regime*, many of the genes have fluctuating values, and attractor cycles can be long. Perturbations of gene values in such networks show strong effects, i.e. these networks are not robust. The networks of the greatest interest for biological systems are thought to lie at the border of the two regimes (the "edge of chaos"), in the so-called *critical regime*. This means that such systems are sufficiently complex, yet robust enough to resemble biological regulation processes. This can also help setting the parameters of networks to be reconstructed.

In the limit (for $n \to \infty$), the regimes are determined by two network parameters: the number of input genes k and the bias p. In detail, the system is chaotic for $k \cdot 2p \cdot (1-p) > 1$ and ordered for $k \cdot 2p \cdot (1-p) < 1$. In the uniform case where $p = 0.5$, the border between the ordered regime and the chaotic regime is at $k = 2$.

Figure 2 shows the so-called critical curve that marks the border between the regimes for different values of p and k (adapted from Aldana 2003). Networks with parameter combinations above the curve are chaotic.

Kauffman's original assumption was that the number of attractors in critical networks with n genes is approximately $\sqrt{n}$ (Kauffman 1993). However, more recent research has shown that the number of attractors grows superpolynomially (i.e. greater than any power law) with n (Samuelsson and Troein 2003). Although it has been shown

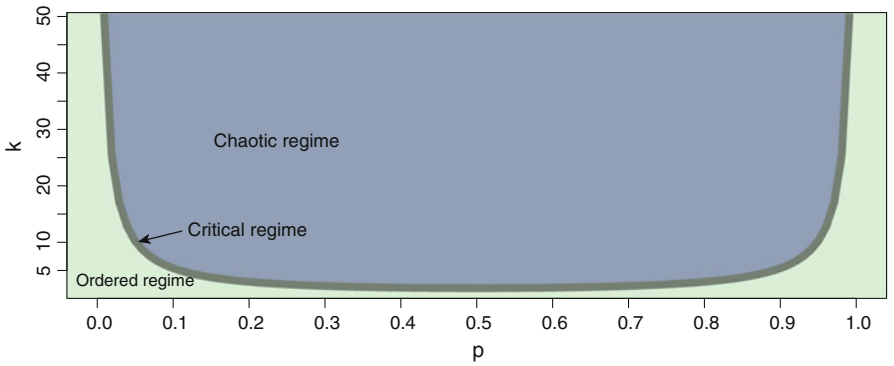

Fig. 2 The critical curve marking the border between the chaotic regime and the ordered regime. To which regime a network belongs depends on the number of input genes of the transition functions k and the bias of the functions p

that the mean number of states in an attractor is superpolynomial in n as well, this is mainly due to a few RBN with very large cycles, while most attractors are rather small (Lynch 1995).

The *BoolNet* R package (Müssel et al. 2010) provides methods to identify and analyze the attractors of synchronous, asynchronous and probabilistic Boolean networks. In the following, we give an introduction to attractor search algorithms implemented in *BoolNet* and demonstrate the use of these algorithms.

2 Synchronous attractor search

A straight-forward algorithm to identify attractors in synchronous Boolean networks starts from a set of start states and repeatedly performs state transitions until an already visited state is reached. Algorithm 1 describes this algorithm in pseudocode. If a state has already been visited, it is either in the basin of attraction of a previously identified attractor, or we have found a new attractor.

Each state s is associated with an attractor number $attractorAssignment(s)$, which is empty at the beginning and indicates the basin of attraction the state belongs to after the search.

If the search reaches an unvisited state s, the algorithm assumes that the state belongs to a yet unidentified attractor. The state's attractor number $attractorAssignment(s)$ is set to the number of the next attractor that will be identified, $currentAttractor$ (that is, the number of already identified attractors plus one). This is done in lines 8–11 of Algorithm 1.

If we come across a previously visited state that has exactly the same attractor number $currentAttractor$, we are in a loop and have identified a new attractor. In this case, the search identifies all states that belong to the attractor and stores them in the attractor list (see lines 12–19 of Algorithm 1).

If the algorithm reaches a previously visited state s that has an attractor number different from $currentAttractor$, the path leads to a previously identified attractor, and thus our assumption of a new attractor is wrong. In this case, we have to step back

Algorithm 1: Synchronous attractor search

Input: A Boolean network with n genes and an n-dimensional state transition function $\mathbf{T}_{sync} : \mathbb{B}^n \to \mathbb{B}^n$
A set of m start states, $\mathcal{S} = \{(s_{11}, \ldots, s_{1n}), \ldots, (s_{m1}, \ldots, s_{mn})\}$

1: $currentAttractor \leftarrow 0$
2: $resultList \leftarrow \emptyset$
3: $attractorAssignment(s) \leftarrow 0$ for all 2^n possible states s
{Mark all states as not assigned to an attractor}
4: **for all** $startState \in \mathcal{S}$ **do**
5: **if** $(attractorAssignment(startState) = 0)$
then {State is not assigned to an attractor}
6: $current \leftarrow startState$
7: $currentAttractor \leftarrow currentAttractor + 1$
{start a new attractor}
8: **while** $(attractorAssignment(current) = 0)$
do {State is not assigned to an attractor}
9: $attractorAssignment(current) \leftarrow currentAttractor$
10: $current \leftarrow \mathbf{T}_{sync}(current)$
11: **end while**
12: **if** $(attractorAssignment(current) = currentAttractor)$
then {A new attractor was found}
13: $attractorStart \leftarrow current$
14: $attractor \leftarrow \emptyset$
15: **repeat** {Identify states in attractor}
16: $attractor \leftarrow attractor \cup \{current\}$
17: $current \leftarrow \mathbf{T}_{sync}(current)$
18: **until** (current = attractorStart)
19: $resultList \leftarrow resultList \cup \{attractor\}$
{Add new attractor to result list}
20: **else** {Correct the attractor assignments of the previous states}
21: $attractorStart \leftarrow current$
22: $current \leftarrow startState$
23: **while** $(current \neq attractorStart)$
do
24: $attractorAssignment(current)$
$\leftarrow attractorAssignment(attractorStart)$
{Assign the current state to the previously identified attractor}
25: $current \leftarrow \mathbf{T}_{sync}(current)$
26: **end while**
27: **end if**
28: **end if**
29: **end for**
30: **return** resultList

to the start state and correct the attractor number $attractorAssignment(s')$ of all states on the path from s' to $attractorAssignment(s)$ (lines 20–27 of Algorithm 1).

To ensure that all attractors in the network are found, an exhaustive search can be performed. This requires using all 2^n states as start states. Due to the exponential complexity of $\mathcal{O}(2^n)$ in both space and runtime, exhaustive search becomes intractable at around 30 genes. By using a predefined or randomly drawn subset of start states, larger networks can be examined in a heuristic way, and the most important attractors with the largest basins of attraction are likely to be found.

In the worst case (for example, if the whole network is one large attractor), the algorithm traverses all 2^n states of a network with n genes even if only a subset of start states is chosen. Thus, the worst-case complexity of the heuristic algorithm is $\mathcal{O}(2^n)$ as well. However, as pointed out previously, large attractors are rare, so that usually larger networks of several hundreds of genes are still tractable.

In the *BoolNet* R package, attractor search is implemented in function `getAttractors()`. This function supports both synchronous and asynchronous search. In the default case, a synchronous search as in Algorithm 1 is conducted. The following example performs an exhaustive attractor search, taking all possible states of the network as input states $\mathcal{S}$. The employed network is included in *BoolNet* and models the mammalian cell cycle (Fauré et al. 2006). It consists of 10 genes and thus has 1,024 possible states.

```
> library("BoolNet")
> data("cellcycle")
> attr <- getAttractors(cellcycle)
> attr
```

```
Attractor 1 is a simple attractor consisting of 1
state(s) and has a basin of 512 state(s):

|--<---------|
V            |
0100010100   |
|            |
V            |
|-->---------|
Genes are encoded in the following order: cycd rb e2f
cyce cyca p27 cdc20 cdh1 ubch10 cycb

Attractor 2 is a simple attractor consisting of 7
state(s) and has a basin of 512 state(s):

|--<---------|
V            |
1001100000   |
|            |
1000100011   |
|            |
1000101011   |
|            |
1000001110   |
|            |
1010000110   |
|            |
1011000100   |
|            |
1011100100   |
|            |
V            |
|-->---------|
```

```
Genes are encoded in the following order: cycd rb e2f
cyce cyca p27 cdc20 cdh1 ubch10 cycb
```

The mammalian cell cycle has two synchronous attractors—one steady-state attractor in the absence of cyclin D, and an attractor with seven states when cyclin D is present. This corresponds to the findings of Fauré et al. (2006).

The attractors and the corresponding basins of attraction can be visualized using `getStateGraph()`. This function plots a transition graph in which the basins of attraction are drawn in different colors, and the attractors are highlighted. The result of

```
> plotStateGraph(attr)
```

is depicted in Fig. 3. The blue basin belongs to the steady-state attractor, and the green basin belongs to the 7-state cycle.

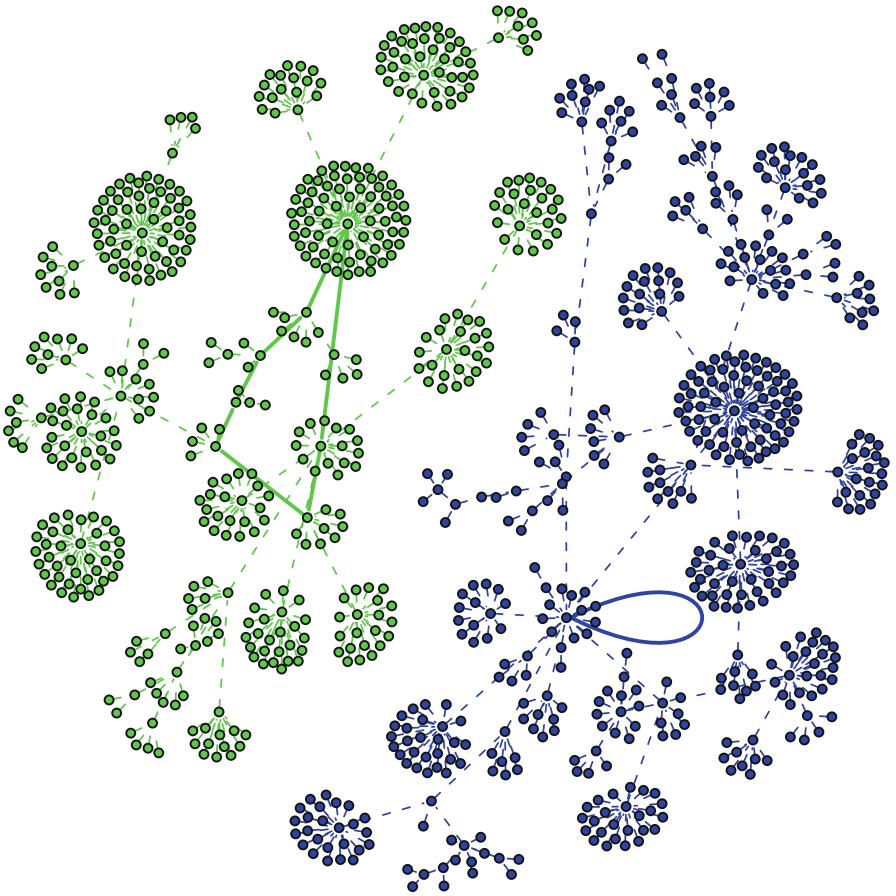

Fig. 3 The state graph of the mammalian cell cycle network. Each *node* represents a state of the network, and each *arrow* is a state transition. The *colors* mark different basins of attraction. Attractors are highlighted using *bold lines* (color figure online)

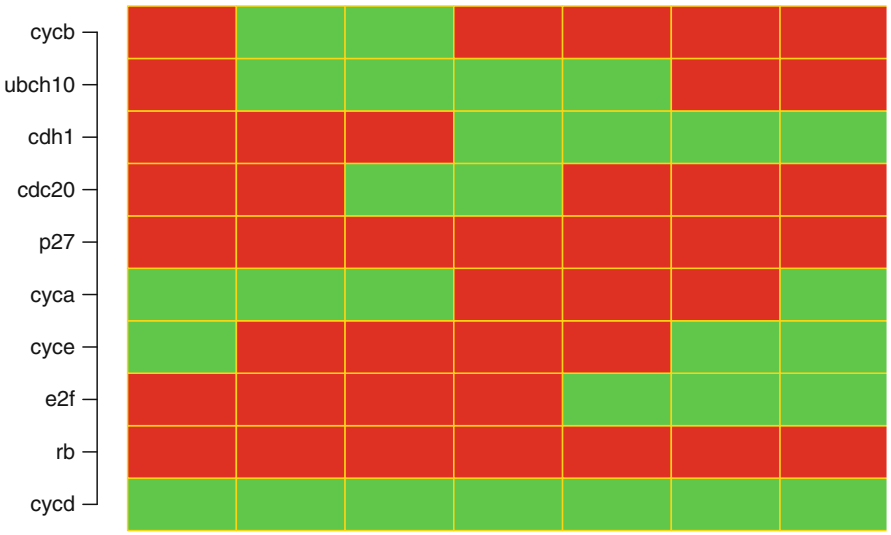

Fig. 4 Visualization of the state changes of the seven-state attractor of the mammalian cell cycle network. The columns of the table represent consecutive states of the attractor. *Green cells* denote an active gene at a certain point of time, whereas *red genes* denote an inactive gene (color figure online)

Instead of exhaustive search, the user can also specify the desired start state(s) for a synchronous attractor search. The algorithm then identifies the attractors to which state transitions from these states lead. The following example calculates successive state transitions beginning at the state where all genes are transcribed:

```
> attr <- getAttractors(cellcycle, method="chosen",
+ startStates=list(rep(1,10)))
```

This again identifies the above attractor with seven states. The attractor can be visualized in a table of expression level changes:

```
> plotAttractors(attr)
```

The resulting plot is given in Fig. 4.

3 Asynchronous attractor search

Asynchronous attractor search is more complex than synchronous attractor search, as there is usually more than one successor state for each state. The algorithm presented here is a new heuristic method that starts with a random walk phase: From a set of randomly chosen start states, a high number of random state transitions is performed to enter an attractor with high probability (see lines 4–7 of Algorithm 2). The algorithm now assumes that the final state is located in a potential attractor and calculates the states of this attractor by determining the forward reachable set of this final state (line 8 of Algorithm 2 and function `ForwardSet()`).

Recall from Sect. 1 that the forward reachable set $FS(\mathbf{s})$ of a state $\mathbf{s}$ consists of all states that can be reached by iterative transitions from the current state, including

Algorithm 2: Asynchronous attractor search

Input: A Boolean network with n genes and n asynchronous state transition functions
$\mathbf{T}^{(i)}_{async} : \mathbb{B}^n \to \mathbb{B}^n$
A number of random transitions r
A set of m start states, $\mathcal{S} = \{(s_{11}, \ldots, s_{1n}), \ldots, (s_{m1}, \ldots, s_{mn})\}$

```
1: resultList ← ∅
2: for all startState ∈ S do
3:    currentState ← startState
4:    for i = 1, ..., r
      do {Perform a random asynchronous state transition on currentState}
5:       i* ← rand(n)
6:       currentState ← T^(i*)_async(currentState)
7:    end for
8:    attractor ← ForwardSet(currentState)
9:    if ValidateAttractor(attractor) then {This is a true attractor}
10:      resultList ← resultList ∪ {attractor}
11:   end if
12: end for
13: return resultList
```

Function `ForwardSet`

Input: A state s for which the forward reachable set is determined

```
1: resultSet ← {s}
2: stack ← {s}
3: repeat
4:    current ← pop(stack)
5:    for i = 1, ..., n
      do {Calculate successor states}
6:       next ← T^(i)_async(current)
7:       if (next ∉ resultSet) then
8:          resultSet ← resultSet ∪ {next}
9:          push(stack, next)
10:      end if
11:   end for
12: until (stack = ∅)
13: return resultSet
```

the state itself. The function `ForwardSet()` constructs the forward reachable set by a depth-first search on all possible state transitions for a current state. Here, the search is described in an iterative manner using a stack, where `pop()` corresponds to taking the top-level element from the stack, and `push()` corresponds to pushing a new element on top of the stack. Each state is only examined once, which is ensured by line 7 of `ForwardSet()`.

The determined states in the set are members of the attractor, provided that the random walk phase succeeded in entering an attractor. To make sure that a true attractor was reached, the set of identified states is validated according to the definition of a complex attractor: An attractor is a set of states such that the forward reachable sets of all states in the set are equal (see, e.g., Garg et al. 2008b). The function `ValidateAttractor()` verifies the definition by comparing the forward reachable sets

Function `ValidateAttractor`

Input: A set of states $\mathcal{S}$ to be validated

1: **for all** $s \in \mathcal{S}$ **do**
2: **if** $(ForwardSet(s) \neq \mathcal{S})$ **then**
3: **return** false
4: **end if**
5: **end for**
6: **return** true

of all identified states to the potential attractor itself—which is the forward reachable set of the final state of the random walk. If any of the sets differ from each other, the potential attractor is rejected, as it is no true attractor. Only the potential attractors that pass the validation are returned from the attractor search (lines 9–11 of Algorithm 2).

The complexity of the algorithm largely depends on the size of the forward reachable sets, i.e. the size of the attractors. The search for the forward reachable set is essentially exponential if states have more than one successor state. If all 2^n states lie in the forward reachable set of a certain state s, the complexity of the function `ForwardSet(s)` is 2^n. Thus, the worst-case complexity of the validation procedure is $\mathcal{O}(m \cdot 2^n \cdot 2^n) = \mathcal{O}(m \cdot 2^{2n})$ for m start states in the improbable case that the whole network is a single complex attractor. Worst-case complexity still is exponential in the size of the attractor, but not in the size of the whole network. In most practical cases, the algorithm will have a reasonable complexity, as attractors are small. This algorithm is also implemented in *BoolNet* and can be called using the command:

```
> attr <- getAttractors(cellcycle, type="asynchronous",
+ method="random", startStates=500)
> attr

Attractor 1 is a simple attractor consisting of 1
state(s):

|--<---------|
V            |
0100010100   |
|            |
V            |
|-->---------|

Genes are encoded in the following order: cycd rb e2f
cyce cyca p27 cdc20 cdh1 ubch10 cycb

Attractor 2 is a complex/loose attractor consisting of
112 state(s) and 338 transition(s):

1011101111 => 1011101110
[...]
1000000000 => 1010000000

Genes are encoded in the following order: cycd rb e2f
cyce cyca p27 cdc20 cdh1 ubch10 cycb
```

The search starts at 500 randomly chosen states and performs 1,000 random state transitions from each of these states. This can be set using the parameter `randomChainLength`.

The function retrieves the previously identified steady-state attractor, which is both a synchronous and an asynchronous attractor of the network. For the complex attractor, the involved transitions are printed. By default, the algorithm tries to avoid self-loops, i.e. transitions that lead to the same state again. This means that self-loop transitions are only allowed if there is no other transition that leads to a different state. If the algorithm should enter self-loops even if transitions to different states are possible, the parameter `avoidSelfLoops` can be set to `FALSE`, resulting in attractors with more state transitions.

As there are multiple possible transitions for each state, complex attractors cannot be visualized in a transition table as above. `plotAttractors()` provides a second visualization that visualizes the transitions among the states in the attractor as a graph:

```
> plotAttractors(attr,subset=2,mode="graph",
  drawLabels=FALSE)
```

plots the 112-state attractor as depicted in Fig. 5. We omit the state labels (i.e. the gene values) due to the high number of states.

4 Probabilistic Boolean networks

Probabilistic Boolean networks were proposed by Shmulevich et al. (2002) to overcome the deterministic rigidity of Boolean networks. Biological uncertainty, experimental noise, or incomplete understanding of a system often necessitate the introduction of uncertainty into the regulatory logic. Probabilistic Boolean networks address this issue by extending the Boolean network model in a way that each target gene can have several candidate functions with associated probabilities.

The dynamic behaviour of a probabilistic Boolean network can be analyzed using Markov chains. A Markov chain simulation does not identify attractor cycles explicitly. Instead, it calculates the probability that a certain state is reached after a predefined number of iterations. States that have a non-zero probability of being reached after a high number of iterations are likely to belong to an attractor.

A probabilistic Boolean network can be viewed as a collection of synchronous Boolean networks assembled by combining all transition function alternatives for the genes. The total number of possible synchronous Boolean networks that can be constructed from a probabilistic Boolean network is $\prod_{i=1}^{n} k_i$. The probability that a certain network N with transition function indices $(j_1, \ldots, j_n)$ is chosen is

$$Pr(\text{Network } N = \left(j_1^*, \ldots, j_n^*\right) \text{ is chosen}) = \prod_{i=1}^{n} p_{i,j_i^*}$$

For each pair of states $\mathbf{y} = (y_1, \ldots, y_n)$ and $\mathbf{z} = (z_1, \ldots, z_n)$, we can determine the set of networks $\mathcal{N}_{yz}$ that yield state $\mathbf{z}$ as the result of a transition from state $\mathbf{y}$. From

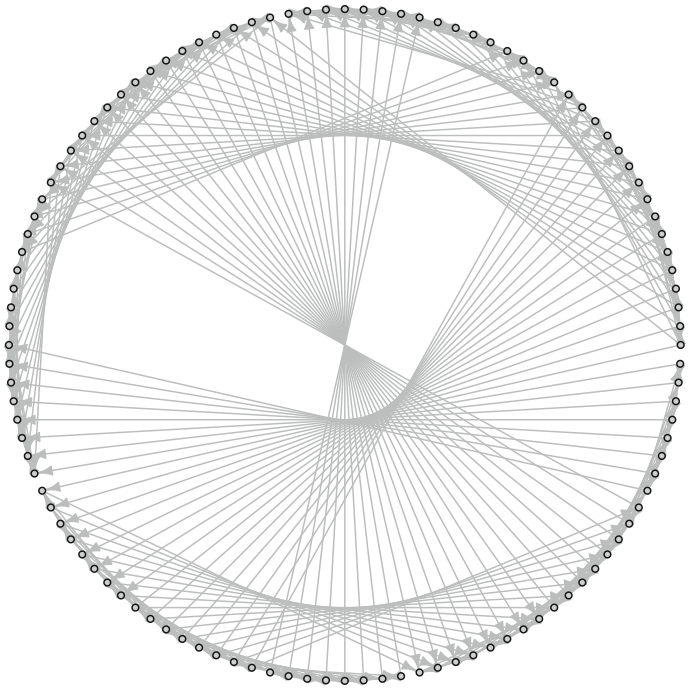

Fig. 5 *Graph* representation of the complex attractor in the mammalian cell cycle network. Each *node* represents a state of the complex attractor, and each *arrow* represents a state transition. The state labels are omitted due to the high number of states

this set, the total probability of a transition from state **y** to state **z** can be calculated:

$$\begin{aligned} Pr(\text{Transition from } \mathbf{y} \text{ to } \mathbf{z}) &= \sum_{N \in \mathcal{N}_{yz}} Pr(\text{Network } N = \left(j_1^*, \ldots, j_n^*\right) \text{ is chosen}) \\ &= \sum_{N \in \mathcal{N}_{yz}} \prod_{i=1}^{n} p_{i,j_i^*} \end{aligned}$$

By collecting the probabilities for all state transitions, a state transition matrix $A \in \mathbb{R}^{2^n \times 2^n}$ can be constructed, where the rows correspond to the initial states, and the columns correspond to the successor states. Each entry of the matrix specifies the probability of a state transition from the state belonging to the corresponding row to the state belonging to the corresponding column.

In the Markov chain simulation, this matrix is multiplied repeatedly with a vector $D^t \in \mathbb{R}^{2^n}$ which holds the probability of each state to be reached after t transitions. The probabilities in D^t sum up to one. The initial vector D^0 determines the probability of a state to be chosen as a start state. Usually, all states have equal initial probabilities. Yet, D^0 can also be used to restrict the search space to certain start states by setting the probabilities for other states to 0. Based on A and the probability vector D^t at

time t, we can compute the probabilities after the next transition (that is, after $t+1$ transitions in total):

$$\begin{aligned} D^{t+1} &= D^t \cdot A \\ &= D^0 \cdot A^{t+1}, \end{aligned}$$

The states in D that have a probability greater than zero after many iterations are called absorbing or irreducible. Once the system enters such a state, it never exits them. The absorbing states correspond to singleton attractors, and the irreducible states correspond to cyclic attractors in Boolean networks. As synchronous Boolean networks are a special case of probabilistic Boolean methods, the Markov chain approach also constitutes an alternative analysis method for synchronous Boolean networks.

The complexity of this algorithm is determined by the number of non-zero elements in the transition matrix, which can be calculated in $\mathcal{O}(\prod_{i=1}^{n} k_i \cdot 2^n)$ by deriving the successor states of all 2^n states for all possible networks. This is also the maximum amount of space needed for the matrix when using a sparse representation. Furthermore, the number of non-zero elements influences the matrix multiplication in further steps. An approach that reduces the number of non-zero elements by calculating the transition matrix only for the states in an attractor was proposed by Garg et al. (2008a). This requires an additional step for the identification of these attractor states which has exponential complexity as well.

BoolNet implements the full Markov chain approach which is outlined above. The following call starts a Markov chain experiment in *BoolNet* with the predefined number of 1,000 iterations on the example probabilistic network described by Shmulevich et al. (2002):

```
> data(examplePBN)
> sim <- markovSimulation(examplePBN)
> sim

States reached at the end of the simulation:
x1 x2 x3 Probability
1  0  0  0        0.15
2  1  1  1        0.85

Probabilities of state transitions in the network:
State     Next state  Probability
000 =>         000          1.0
001 =>         110          1.0
010 =>         110          1.0
011 =>         000          0.2
011 =>         100          0.3
011 =>         001          0.2
011 =>         101          0.3
100 =>         010          1.0
101 =>         110          0.5
101 =>         111          0.5
```

```
110 =>          100          0.5
110 =>          101          0.5
111 =>          111          1.0
```

The first table of the output shows the states that are reached at the end of the simulation. This includes only those states that have a non-zero probability of being reached after the 1,000 iterations. The second table is a transition table annotated with transition probabilities. This table can be suppressed by the parameter `returnTable=FALSE`.

5 Summary

The identification and analysis of attractors in Boolean networks is crucial for understanding their dynamics and the associated biological functions.

Boolean networks have demonstrated their ability to capture the behaviour of many biological systems. For example, Orlando et al. (2008) analyze the attractors of a Boolean model to validate their hypotheses on the transcription processes during the cell cycle of budding yeast. Sahin et al. (2009) identified a potential target protein in breast cancer cells using artificial knockdowns in Boolean networks. Calzone et al. (2010) construct a Boolean model of cell fate decision and derive specific predictions regarding cross-talks between three pathways. Meanwhile, theoretical aspects of the Boolean model have been studied extensively, including analyses of biologically relevant functions (Kauffman et al. 2004) and the impact of perturbations (Xiao and Dougherty 2007).

This tutorial provides an overview of the implementation and usage of attractor search algorithms in the *BoolNet* R-package. It covers attractor search in synchronous, asynchronous and probabilistic Boolean networks.

While this paper discusses the identification of attractors, Boolean network analysis covers many more aspects, such as modeling of networks, reconstructing networks from time series measurements, or robustness assessment in perturbation experiments. A practical overview of these topics including many examples of the corresponding methods in *BoolNet* is given in the *BoolNet* package vignette. The vignette is included in the package and can be displayed by calling

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

in an R prompt. Furthermore, the Statistical Computing 2010 tutorial and introduction are available at http://www.statistical-computing.de/Reisensburg2010.

Acknowledgments This project was partly funded by the German Federal Ministry of Education and Research (BMBF) within the framework of the program of medical genome research (PaCa-Net; project ID PKB-01GS08) and Gerontosys (Forschungskern SyStaR). It is also supported by the Graduate School of Mathematical Analysis of Evolution, Information, and Complexity at the University of Ulm (CM, HAK), and by the International Graduate School in Molecular Medicine Ulm, funded by the Excellence Initiative of the German Federal and State Governments (HAK).

References

Albert R, Othmer H (2003) The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. J Theor Biol 223(1):1–18

Aldana M (2003) Boolean dynamics of networks with scale-free topology. Physica D: Nonlinear Phenomena 185(1):45–66

Bornholdt S (2005) Systems biology. Less is more in modeling large genetic networks. Science 310(5747):449–451

Bornholdt S (2008) Boolean network models of cellular regulation: prospects and limitations. J R Soc Interface 5(Suppl. 1):S85–S94

Calzone L, Tournier L, Fourquet S, Thieffry D, Zhivotovsky B, Barillot E, Zinovyev A (2010) Mathematical modelling of cell-fate decision in response to death receptor engagement. PLoS Comput Biol 6(3):e1000702

de Jong H (2002) Modeling and simulation of genetic regulatory systems: a literature review. J Comput Biol 9(1):67–103

Dojer N, Gambin A, Mizera A, Wilczyński B, Tiuryn J (2006) Applying dynamic Bayesian networks to perturbed gene expression data. BMC Bioinform 7:249

Fauré A, Naldi A, Chaouiya C, Thieffry D (2006) Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. Bioinformatics 22(14):e124–e131

Garg A, Banerjee D, De Micheli G (2008) Implicit methods for probabilistic modeling of gene regulatory networks. In: Proceedings of the 30th annual international IEEE EMBS conference, pp 4621–4627

Garg A, Di Cara A, Xenarios I, Mendoza L, De Micheli G (2008) Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics 24(17):1917–1925

Glass L, Kauffman SA (1973) The logical analysis of continuous, non-linear biochemical networks. J Theor Biol 39(1):103–129

Goodwin B (1963) Temporal organization in cells: a dynamic theory of cellular control processes. Academic Press, London

Grzegorczyk M, Husmeier D, Rahnenführer J (2010) Modelling non-stationary dynamic gene regulatory processes with the BGM model. Comput Stat 26(2):199–218

Harvey I, Bossomaier T (1997) Time out of joint: attractors in asynchronous random Boolean networks. In: Proceedings of the forth European conference on artificial life, pp 67–75

Huang S (1999) Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery. J Mol Med 77(6):469–480

Kauffman SA (1969) Metabolic stability and epigensis in randomly constructed genetic nets. J Theor Biol 22(3):437–467

Kauffman SA (1993) The origins of order: self-organization and selection in evolution. Oxford University Press, Oxford

Kauffman S, Peterson C, Samuelsson B, Troein C (2004) Genetic networks with canalyzing Boolean rules are always stable. PNAS 101(49):17,102–17,107

Lähdesmäki H, Shmulevich I, Yli-Harja O (2003) On learning gene regulatory networks under the Boolean network model. Mach Learn 52(1–2):147–167

Li F, Long T, Ouyang Q, Tang C (2004) The yeast cell-cycle network is robustly designed. PNAS 101:4781–4786

Liang S, Fuhrman S, Somogyi R (1998) REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. Pac Symp Biocomput 3:18–29

Lynch J (1995) On the threshold of chaos in random Boolean cellular automata. Random Struct Algorithms 6(2–3):239–260

Müssel C, Hopfensitz M, Kestler HA (2010) BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. Bioinformatics 26(10):1378–1380

Orlando DA, Lin CY, Bernard A, Wang JY, Socolar JES, Iversen ES, Hartemink AJ, Haase SB (2008) Global control of cell-cycle transcription by coupled CDK and network oscillators. Nature 453(7197):944–947

Sahin O, Fröhlich H, Löbke C, Korf U, Burmester S, Majety M, Mattern J, Schupp I, Chaouiya C, Thieffry D, Poustka A, Wiemann S, Beissbarth T, Arlt D (2009) Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance. BMC Syst Biol 3(1):1

Samuelsson B, Troein C (2003) Superpolynomial growth in the number of attractors in Kauffman networks. Phys Rev Lett 90(9):098701

Shmulevich I, Dougherty ER, Kim S, Zhang W (2002) Probabilistic Boolean networks: a rule-based uncertainty model for gene-regulatory networks. Bioinformatics 18(2):261–274

Socolar JE, Kauffman SA (2003) Scaling in ordered and critical random Boolean networks. Phys Rev Lett 90(6):068–702

Thomas R (1991) Regulatory networks seen as asynchronous automata: a logical description. J Theor Biol 153(1):1–23

Wawra C, Kühl M, Kestler HA (2007) Extended analyses of the Wnt/β-catenin pathway: Robustness and oscillatory behaviour. FEBS Lett 581(21):4043–4048

Xiao Y, Dougherty ER (2007) The impact of function perturbations in Boolean networks. Bioinformatics 23(10):1265–1273

Zhou D, Müssel C, Lausser L, Hopfensitz M, Kühl M, Kestler HA (2009) Boolean networks for modeling and analysis of gene regulation. Ulmer Informatik-Bericht 2009–2010, Ulm University