Algorithms and methods of the `BoolNet` R–package

Christoph Müssel, Martin Hopfensitz, Hans A. Kestler

Abstract

This document describes the algorithms and methods that were developed or partially adapted for the `BoolNet` package. This includes algorithms for attractor search and binarization.

1 Synchronous attractor search

Algorithm 1 describes a straight-forward algorithm to identify attractors in synchronous Boolean networks. This algorithm starts from a set of start states (which consists of all states in case of exhaustive search, or of a subset of states for heuristic search) and repeatedly performs synchronous state transitions. To determine whether a state has already been processed, a number $attractorAssignment(s)$ corresponding to the next attractor to be identified is assigned to the states when they are reached. This number later corresponds to the basin of attraction the state belongs to. If the algorithm reaches a state that has already been processed and is in the basin of attraction of the next unidentified attractor, this state is part of this attractor. Further states that belong to the attractor can now be determined by repeatedly performing state transitions until the state is reached again.

2 Asynchronous attractor search

Asynchronous attractor search starts with a random walk phase: A high number of random state transitions is performed to enter an attractor with high probability. 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.

The forward reachable set of a state consists of all states that can be reached by iterative transitions from the current state, including the state itself. It is constructed by the function `ForwardSet()` displayed below. This function performs a depth-first search on all possible state transitions for a current state. Here, `pop()` and `push()` correspond to the commonly used stack operations of taking the top-level element from the stack and pushing a new element on top of the stack. Successor states are only examined if they have not yet been added to the result set.

If the random walk phase lead to an attractor, the forward reachable set of the final state is a (complex or steady-state) attractor. However, it is possible that

Algorithm 1: Synchronous attractor search

```
Input: A Boolean network with $m$ genes and an $m$-dimensional
       transition function $f : \mathbb{B}^m \rightarrow \mathbb{B}^m$
       A set of $n$ start states, $\mathcal{S} = \{(s_{11}, \dots, s_{1m}), \dots, (s_{n1}, \dots, s_{nm})\}$
  $currentAttractor \leftarrow 0$
  $resultList \leftarrow \emptyset$
  $attractorAssignment(s) \leftarrow 0$ for all $2^m$ possible states $s$
  {Mark all states as not assigned to an attractor}
  for all $startState \in \mathcal{S}$ do
    if ($attractorAssignment(startState) = 0$)
    then {State is not assigned to an attractor}
      $current \leftarrow startState$
      $currentAttractor \leftarrow currentAttractor + 1$
      {start a new attractor}
      while ($attractorAssignment(current) = 0$)
      do {State is not assigned to an attractor}
        $attractorAssignment(current) \leftarrow currentAttractor$
        $current \leftarrow f(current)$
      end while
      if ($attractorAssignment(current) = currentAttractor$)
      then {A new attractor was found}
        $attractorStart \leftarrow current$
        $attractor \leftarrow \emptyset$
        repeat {Identify states in attractor}
          $attractor \leftarrow attractor \cup \{current\}$
          $current \leftarrow f(current)$
        until (current = attractorStart)
        $resultList \leftarrow resultList \cup \{attractor\}$
        {Add new attractor to result list}
      else {Correct the attractor assignments of the previous states}
        $attractorStart \leftarrow current$
        $current \leftarrow startState$
        while ($current \neq attractorStart$)
        do
          $attractorAssignment(current)$
          $\leftarrow attractorAssignment(attractorStart)$
          {Assign the current state to the previously identified attractor}
          $current \leftarrow f(current)$
        end while
      end if
    end if
  end for
  return resultList
```

no attractor was attained in this phase. Thus, a validation step is needed to ensure that only true attractors are found. By definition, an attractor is a set of states such that the forward reachable sets of all states in the set are equal

Algorithm 2: Asynchronous attractor search

```
Input: A Boolean network with $m$ genes and $m$ transition functions
       $f_i : \mathbb{B}^m \rightarrow \mathbb{B}^m$ changing a single gene
       A number of random transitions $r$
       A set of $n$ start states, $\mathcal{S} = \{(s_{11}, \ldots, s_{1m}), \ldots, (s_{n1}, \ldots, s_{nm})\}$
  $resultList \leftarrow \emptyset$
  for all $startState \in \mathcal{S}$ do
    $currentState \leftarrow startState$
    for $i = 1, \ldots, r$ do
      Perform a random asynchronous state transition on $currentState$
    end for
    $attractor \leftarrow ForwardSet(currentState)$
    if $ValidateAttractor(attractor)$ then {This is a true attractor}
      $resultList \leftarrow resultList \cup \{attractor\}$
    end if
  end for
  return resultList
```

Function `ForwardSet`

```
Input: A state $s$ for which the forward reachable set is determined
  $resultSet \leftarrow \{s\}$
  $stack \leftarrow \{s\}$
  repeat
    $current \leftarrow pop(stack)$
    for $i = 1, \ldots, m$
    do {Calculate successor states}
      if $(f_i(current) \notin resultSet)$ then
        $resultSet \leftarrow resultSet \cup \{f_i(current)\}$
        $push(stack, f_i(current))$
      end if
    end for
  until $(stack = \emptyset)$
  return resultSet
```

Function `ValidateAttractor`

```
Input: A set of states $\mathcal{S}$ to be validated
  for all $s \in \mathcal{S}$ do
    if $(ForwardSet(s) \neq \mathcal{S})$ then
      return false
    end if
  end for
  return true
```

(see, e.g., [3]). Consequently, the function `ValidateAttractor()` compares the forward reachable sets of all states in the potential attractor to the potential attractor itself. As soon as a different set is found, the function states that the supplied set of states is no true attractor. Only the potential attractors that pass the validation are returned from the attractor search.

3 Scan statistics

Scan statistics have been developed to identify unusually large clusters of measurements [4]. They are used to decide whether large accumulations of measurements are unlikely to have arisen by chance if measurements are distributed independently and at random.

A scanning window of fixed size is shifted across the measurements to decide at each of the window positions whether there is an unusual accumulation of measurements. The window size is given as a fraction of the whole scanning space w. The decision if an unusual accumulation of measurements occurred is based on a probability $P(k; N, w)$: Given N measurements in a fixed range, $P(k; N, w)$ is the probability to find as many as k measurements in a range (window) of size w at random. Glaz et al. [4] give an approximation for $P(k; N, w)$:

$$P(k; N, w) = (kw^{-1} - N - 1) \cdot b \cdot (k; N, w) + 2 \cdot G_b(k; N, w),$$

where

$$\begin{aligned} b(k; N, w) &= \binom{N}{k} w^k (1-w)^{N-k} \\ G_b(k; N, w) &= \sum_{i=k}^{N} b(i; N, w) \end{aligned}$$

This approximation is exact for $P < 0.1$ and even larger values [4].

3.1 Binarization based on scan statistics

We developed a novel binarization method that is based on threshold determination using scan statistics and additionally provides a measure of threshold validity. The main idea is to search for at least one cluster in the measurements whose probability P is lower than a specified significance level α. If there is no cluster with the required significance, then no reliable binarization can be determined since the data is distributed more or less uniformly. Based on the detected clusters with a high significance (i.e., a low value of P), the binarization is performed in a way that the points within a cluster are assigned to the same binary value.

In detail, the binarization approach is as follows: As input parameters, the algorithm requires the real-valued data G, the significance level α, and the scanning window size w which is supplied as a fraction of the range of the whole scanning space. In a first step, the real-valued data G is sorted increasingly. Then, the size of the scanning window is determined based on w and the range of the values in the sorted measurements S:

$$windowSize = (\max S - \min S) \cdot w.$$

The scanning window SW is shifted across the data so that the left margins of the scanning window are the values in S.

$$SW_i = [S_i, S_i + windowSize]\,,\; 1 \leq i \leq N.$$

At each position of the scanning window, the number of measurements k that are enclosed by the window is calculated, and the probability $P(k; N, w)$ is determined. If P is smaller than the given significance level α, the position of the scanning window is stored as a solution. If a previously identified solution $s \in solution$ overlaps with the scanning window, the window is joined with this solution:

$$solution \leftarrow (solution \setminus \{s\}) \cup \{SW_i \cup s\}$$

In this way, intervals larger than the size of the scanning window can be stored in *solution*. If a cluster with $P \leq \alpha$ was found, the binarization is reliable ($reject \leftarrow false$). If no cluster with $P \leq \alpha$ was found for $1 \leq i \leq N$, we successively increase α until a solution with the new significance level that can be used for a binarization is found. However, the binarization based on this cluster is considered as unreliable ($reject \leftarrow true$). The subsequent binarization is performed based on the interval that encloses the window with the smallest P that is stored in *solution*. The boundaries of this window form two thresholds, from which the value that results in more balanced groups is taken for binarization.

Algorithm 5: Scan statistics binarization

```
Input: The real-valued data $G$
       A significance level $\alpha$
       A scanning window size (fraction) $w$
 $S \leftarrow sort(G)$
 $N \leftarrow |G|$
 $windowSize \leftarrow (\max S - \min S) \cdot w$
 $solution \leftarrow \emptyset$
 for $i = 1, \ldots, N$ do
   $SW_i \leftarrow [S_i, S_i + windowSize]$
   $k \leftarrow$ number measurements enclosed by $SW_i$
   if $P(k; N, w) \leq \alpha$ then
     if $(\exists s \in solution : s \cap SW_i \neq \emptyset)$ then
       $solution \leftarrow (solution \setminus \{s\}) \cup \{SW_i \cup s\}$
     else
       $solution \leftarrow solution \cup \{SW_i\}$
     end if
   end if
 end for
 if $solution = \emptyset$ then
   search solution with a higher $\alpha$
   $reject \leftarrow true$
 else
   $reject \leftarrow false$
 end if
 $bestWindow \leftarrow s \in solution$ containing the window with the smallest $P$
 $greaterSet \leftarrow$ number of measurements $> bestWindow$
 $smallerSet \leftarrow$ number of measurements $< bestWindow$
 if $greaterSet \leq smallerSet$ then
   $threshold \leftarrow \min bestWindow$
 else
   $threshold \leftarrow \max bestWindow$
 end if
 $B \leftarrow$ binarize $G$ based on $threshold$
 return $(B, reject)$
```

4 Computer-intensive tests for network properties

The `BoolNet` package introduces a generic facility to perform computer-intensive tests for the identification of specific properties of biological networks. This section describes the methodology of these tests.

The core of the test method is a *test function* f which is responsible for calculating a test statistic and extracting the desired properties from the network. This function is supplied with a network *net* to test, a flag *accumulate*, and a list of further function-specific parameters. *accumulate* determines the type of the result: If it is *true*, the function should calculate a test statistic summarizing the desired network property in a single value. If *accumulate* is $false$, a sample of values in form of a vector can be returned without summarizing.

The computer-intensive testing process is as follows (for a detailed introduction see e.g. [2]): At first, m randomly generated networks net_i are created. Each of these networks has the same number of genes and transition functions of the same arity as the original network. For the original network and all random networks, the value of the test function is determined.

The standard method of processing the results of the test functions is to perform a significance test. In this case, the test functions are required to return a single-value test statistic as described above. The results from all random networks then represent the distribution of the test statistic in the random case. The hypotheses of the tests are:
H_0: The biological network exhibits a test statistic value less than or equal to the value of a randomly created network.
H_1: The test statistic of the biological network is greater than the value of a randomly created network.
The fraction of test function values on the random networks that are greater than the test function value of the biological network, i.e. $f(net_i,\ldots) > f(originalNet,\ldots)$, can be used as a p-value. The result of such a significance test can be visualized by a histogram of the random values with vertical lines for the significance level and the p-value.

The second integrated processing method compares empirical distributions of values by calculating the Kullback-Leibler distances between the distributions of the supplied network and each of the random networks [1]. For this purpose, the test function is required to return a sample of values, i.e. *accumulate is false*. Depending on the types of measurements, it may be necessary to perform a binning on the samples for discretization. The Kullback-Leibler distance for a set of events E and two samples X and Y sampled from these events is

$$KL(X,Y) = \sum_{e \in E} \hat{p}_X(e) \cdot \log \frac{\hat{p}_X(e)}{\hat{p}_Y(e)}$$

$\hat{p}_X(e)$ and $\hat{p}_Y(e)$ are estimates of the probabilities of event e in the distributions of X and Y respectively. The Kullback-Leibler distances for all random networks can plotted in a histogram. High distances suggest that there is a significant difference between the random networks and the biological network.

Algorithm 6 shows the simplified testing process.

Algorithm 6: Computer-intensive test for network properties

```
Input: A Boolean network $originalNet$
       A number of random networks $m$
       A test function $f(network, accumulate, params)$
       A vector of additional test function parameters $p_1, \dots, p_k$
       A significance level $\alpha$
       A flag $kl$ specifying if the Kullback-Leibler distance should be
       used for summarization
  Generate $m$ random networks $net_1, \dots, net_m$
  $t_o \leftarrow f(originalNet, \neg kl, p_1, \dots, p_k)$
  for $i \in 1, \dots, m$ do
    $t_i \leftarrow f(net_i, \neg kl, p_1, \dots, p_k)$
  end for
  if (kl) then
    for $i \in 1, \dots, m$ do
      $d_i \leftarrow KL(t_o, t_i)$
    end for
    Plot a histogram $hist$ of $d_i,\ i \in 1, \dots, m$
    return $(hist)$
  else
    $p \leftarrow \frac{|\{t_i | t_i > t_0\}|}{m}$
    $significant \leftarrow (p \leq \alpha)$
    Plot a histogram $hist$ of $t_i,\ i \in 1, \dots, m$, with a vertical line for $t_o$
    return $(hist, p, significant)$
  end if
```

4.1 Integrated test functions

BoolNet includes two predefined test functions that define test statistics for synchronous Boolean networks and can be used with the described testing facility:

TestIndegree builds a state transition graph G from the network and calculates the in-degrees d_j of for each the states, i.e. the number of state transitions leading to state j. If required by the testing facility, the in-degrees are summarized using the Gini index, which is 0 if all states have an equal in-degree of 1, and 1 if all transitions lead to one state. For more details, see Algorithm 7.

TestAttractorRobustness creates c perturbed copies of the supplied network net. It identifies the set of synchronous attractors A_o in net and then counts the numbers of attractors in A_o, f_j, that can be found in each of the c copies. Depending on *accumulate*, the function either returns the overall mean percentage of attractors found in all copies, or a percentage of found attractors for each of the copies. The test function is summarized in Algorithm 8.

Function `TestIndegree`

```
Input: A network $net$
       A flag $accumulate$
  Build a graph $G = (V, E)$ in which the vertices $V = \{v_1, \dots, v_l\}$ are the
  states of $net$ and the edges $E$ are the state transitions
  for $j = 1, \dots, l$
  do {Calculate in-degrees}
    $d_j \leftarrow |\{(v*, v_j) \in E\}|$
  end for
  if ($accumulate$)
  then {Calculate Gini index}
    $D^* \leftarrow sort(d_1, \dots, d_l)$
    return  $Gini(D^*) = \frac{2 \cdot \sum_{j=1}^{l} i \cdot d^*_{(j)} - (l+1) \cdot \sum_{j=1}^{l} d^*_{(j)}}{l \cdot \sum_{j=1}^{l} d^*_{(j)}}$
  else
    return  $(d_1, \dots, d_l)$
  end if
```

Function `TestAttractorRobustness`

```
Input: A network $net$
       A flag $accumulate$
       A number of perturbed copies $c$
  $A_o \leftarrow$ synchronous attractors $\{a_1, \dots, a_s\}$ of $net$
  for $j = 1, \dots, c$
  do {Find attractors in perturbed copies}
    $net^*_j \leftarrow$ perturbed copy of $net$
    $A^*_j \leftarrow$ synchronous attractors of $net^*_j$
    $f_j \leftarrow \left|A_o \cap A^*_j\right|$
  end for
  if ($accumulate$)
  then {Calculate overall percentage of found attractors}
    return  $\frac{1}{c \cdot s} \sum_{j=1}^{c} f_j \cdot 100$
  else
    return  $(\frac{f_1 \cdot 100}{s}, \dots, \frac{f_c \cdot 100}{s})$
  end if
```

References

[1] T. M. Cover and J. A. Thomas. *Information Theory.* Wiley, 1991.

[2] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, 1994.

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

[4] J. Glaz, J. Naus, and S. Wallenstein. *Scan Statistics.* Springer, 2001.