

# Package ‘pastboon’

July 23, 2024

**Type** Package

**Title** Simulation of Parameterized Stochastic Boolean Networks

**Version** 0.1.0

**Description** Applying stochastic noise to Boolean networks is a useful approach for representing the effects of various perturbing stimuli on complex systems. A number of methods have been developed to control noise effects on Boolean networks using parameters integrated into the update rules. This package provides functions to examine three such methods: BNp (Boolean network with perturbations), described by Trairatphisan et al. (2013) <[doi:10.1186/1478-811X-11-46](https://doi.org/10.1186/1478-811X-11-46)>, SDDS (stochastic discrete dynamical systems), proposed by Murrugarra et al. (2012) <[doi:10.1186/1687-4153-2012-5](https://doi.org/10.1186/1687-4153-2012-5)>, and PEW (Boolean network with probabilistic edge weights), presented by Deritei et al. (2022) <[doi:10.1371/journal.pcbi.1010536](https://doi.org/10.1371/journal.pcbi.1010536)>. This package includes source code derived from the 'BoolNet' package, which is licensed under the Artistic License 2.0.

**Author** Mohammad Taheri-Ledari [aut, cre, cph]  
(<<https://orcid.org/0009-0007-9132-077X>>),  
Kaveh Kavousi [ctb] (<<https://orcid.org/0000-0002-1906-3912>>),  
Sayed-Amir Marashi [ctb] (<<https://orcid.org/0000-0001-9801-7449>>),  
Authors of BoolNet [ctb] (Original authors of the BoolNet package),  
Troy D. Hanson [ctb] (contributed uthash macros)

**Maintainer** Mohammad Taheri-Ledari <[mo.taheri@ut.ac.ir](mailto:mo.taheri@ut.ac.ir)>

**Depends** R (>= 3.5.0)

**Suggests** BoolNet

**License** Artistic-2.0

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** yes

**BugReports** <https://github.com/taherimo/pastboon/issues>

**Repository** CRAN

**Date/Publication** 2024-07-23 00:30:06 UTC

## Contents

pastboon-package . . . . .	2
calc_convergence_time . . . . .	3
calc_node_activities . . . . .	5
count_pairwise_trans . . . . .	7
extract_edges . . . . .	10
get_reached_states . . . . .	11
lac_operon_net . . . . .	14

<b>Index</b>	<b>15</b>
--------------	-----------

---

pastboon-package	<i>Simulation of Parameterized Stochastic Boolean Networks</i>
------------------	--

---

## Description

Applying stochastic noise to Boolean networks is a useful approach for representing the effects of various perturbing stimuli on complex systems. A number of methods have been developed to control noise effects on Boolean networks using parameters integrated into the update rules. This package provides functions to simulate and analyze three such methods: *Boolean network with perturbations* (BNp), described by Trairatphisan et al., *stochastic discrete dynamical systems* (SDDS), proposed by Murrugarra et al., and *Boolean network with probabilistic edge weights* (PEW), presented by Deritei et al. The package includes source code derived from the BoolNet package, which is licensed under the Artistic License 2.0.

## Details

Applying perturbations to a standard deterministic Boolean network involves altering its update rules. Manipulating the logical functions usually requires a thorough understanding of the reasoning behind the Boolean equations and may lead to a loss of the network's main functional characteristics, which often need to be preserved. An alternative approach to perturbing a Boolean network is to introduce stochastic noise and control its effect through a set of parameters integrated into the logical functions. This approach offers the advantage of allowing partial activation or inhibition of nodes.

In pastboon, three parameterization methods are implemented to control the stochastic noise effect on Boolean networks:

- BNp, *Boolean network with perturbations* (Trairatphisan et al.)
- SDDS, *Stochastic discrete dynamical systems* (Murrugarra et al.)
- PEW, *Boolean network with probabilistic edge weights* (Deritei et al.)

Given a Boolean network, its parameterization method, and the parameter values, useful insights can be gained from network simulations using the functions provided in this package. Node activity trajectories (the probability of each node being active at each time-step) can be calculated using `calc_node_activities`. Once node activity trajectories are available, the time-step at which the network reaches a steady-state distribution can be estimated using `calc_convergence_time`. Additionally, the states reached after starting a Boolean network from a given set of initial states can be

sampled over specified time-steps using `get_reached_states`. The number of pairwise transitions between a given set of states can be obtained using `count_pairwise_trans`. Finally, the edges of a Boolean network can be extracted using `extract_edges`.

This package includes source code derived from the BoolNet package, which is licensed under the Artistic License 2.0. Specifically, the C code for simulating Boolean networks and its R interface code were initially taken from the BoolNet package but have been substantially altered (particularly the C code) to meet our purposes.

### Author(s)

Mohammad Taheri-Ledari [aut, cre, cph] <mo.taheri@ut.ac.ir>

Kaveh Kavousi [ctb]

Sayed-Amir Marashi [ctb]

Authors of BoolNet [ctb]

Troy D. Hanson [ctb]

### References

Trairatphisan, P., Mizera, A., Pang, J., Tantar, A. A., Schneider, J., & Sauter, T. (2013). Recent development and biomedical applications of probabilistic Boolean networks. *Cell communication and signaling*, 11, 1-25.

Murrugarra, D., Veliz-Cuba, A., Aguilar, B., Arat, S., & Laubenbacher, R. (2012). Modeling stochasticity and variability in gene regulatory networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2012, 1-11.

Deritei, D., Kunšič, N., & Csermely, P. (2022). Probabilistic edge weights fine-tune Boolean network dynamics. *PLoS Computational Biology*, 18(10), e1010536.

Müssel, C., Hopfensitz, M., & Kestler, H. A. (2010). BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10), 1378-1380.

---

calc\_convergence\_time *Calculate convergence time-step for node activities*

---

### Description

Given node activity trajectories for a set of variables `node_act`, this function calculates the time-step from which the changes in all the curves are below `threshold` for `window_size` consecutive time-steps.

### Usage

```
calc_convergence_time(node_act, threshold, window_size = 1)
```

## Arguments

node_act	A matrix describing node activities over consecutive time-steps, where rows represent time-steps and columns represent nodes. It is the output of <a href="#">calc_node_activities</a> .
threshold	A value determining the maximum allowable change in node activities to decide if they have converged.
window_size	The number of consecutive time-steps for which the node activity curves must remain stable (i.e., changes below threshold) to be considered converged. The default is 1.

## Details

The function checks if the changes in all node activity curves are less than threshold for window\_size consecutive time-steps. If this condition is met, the node activity curves are considered to have converged to their stable values, and the convergence time-step (the starting point of the window) is returned. Since node activities represent marginal probabilities of the nodes being active at each time-step, convergence indicates that the steady-state distribution of the corresponding Boolean network has been reached, meaning that the probability of being in each state of the network no longer changes significantly.

## Value

The time-step at which convergence occurs. If no convergence is detected, NA is returned.

## Examples

```
# Load the example network
data(lac_operon_net)

# Define parameters for the SDDS method
props <- rep(0.95, length(lac_operon_net$genes))
params <- list(p00 = props, p01 = props, p10 = props, p11 = props)

# Get node activities after simulation using the SDDS method
node_act <- calc_node_activities(lac_operon_net, method = "SDDS", params = params,
  steps = 100, repeats = 10000)

# Calculate the convergence time
convergence_time <- calc_convergence_time(node_act, threshold = 0.01)

# Print the convergence time
print(convergence_time)
```

---

calc\_node\_activities *Calculate activity rate for each node*

---

### Description

Calculates the activity rate of the nodes (i.e., the number of times a node is active, i.e., ON, divided by the number of repeats) for a specified number of time-steps.

### Usage

```
calc_node_activities(net, method = c("BNp", "SDDS", "PEW"), params, steps,
                    repeats = 1000, initial_prob = NULL, last_step = FALSE,
                    asynchronous = TRUE, update_prob = NULL)
```

### Arguments

net	A network structure of the class BooleanNetwork from the BoolNet package.
method	The parameterization method to be used. Options are: <ul style="list-style-type: none"> <li>"BNp": Boolean network with perturbations.</li> <li>"SDDS": Stochastic discrete dynamical systems.</li> <li>"PEW": Boolean network with probabilistic edge weights.</li> </ul> Each method requires a different format for the params argument.
params	The parameter values depending on method: <ul style="list-style-type: none"> <li>For method = "BNp", a single vector of probabilities, equal in length to the number of network nodes.</li> <li>For method = "SDDS", a list of four equal-length vectors of probabilities: p00, p01, p10, and p11, each equal in length to the number of network nodes.</li> <li>For method = "PEW", a list of two equal-length vectors of probabilities: p_on and p_off, each as long as the number of network edges, ordered according to <a href="#">extract_edges</a>.</li> </ul>
steps	The number of time-steps (non-negative integer) to simulate the network.
repeats	The number of repeats (positive integer).
initial_prob	The probability that each of the nodes is ON (1) in the initial state (time-step 0). If NULL (default), 0.5 is used as the probability for all nodes, meaning the initial state is randomly chosen based on a uniform distribution.
last_step	If TRUE, only the node activity rates for the last time-step are returned. Otherwise, the entire activity rate trajectory of the nodes is returned.
asynchronous	If TRUE, the asynchronous update scheme is used, where a single node is updated at each time-step. In this case, update_prob indicates update probabilities. If FALSE, the synchronous update scheme is utilized.
update_prob	The probability of updating each variable (node) in each time-step when asynchronous = TRUE. If NULL (default), nodes are updated randomly based on a uniform distribution. If asynchronous = FALSE, this argument is ignored.





**Usage**

```
count_pairwise_trans(net, method = c("BNp", "SDDS", "PEW"), params, states,
                    steps = 1, repeats = 1000, asynchronous = TRUE,
                    update_prob = NULL)
```

**Arguments**

net	A network structure of the class BooleanNetwork from the BoolNet package.
method	The parameterization method to be used. Options are: <ul style="list-style-type: none"> <li>• "BNp": Boolean network with perturbations.</li> <li>• "SDDS": Stochastic discrete dynamical systems.</li> <li>• "PEW": Boolean network with probabilistic edge weights.</li> </ul> Each method requires a different format for the params argument.
params	The parameter values depending on method: <ul style="list-style-type: none"> <li>• For method = "BNp", a single vector of probabilities, equal in length to the number of network nodes.</li> <li>• For method = "SDDS", a list of four equal-length vectors of probabilities: p00, p01, p10, and p11, each equal in length to the number of network nodes.</li> <li>• For method = "PEW", a list of two equal-length vectors of probabilities: p_on and p_off, each as long as the number of network edges, ordered according to <a href="#">extract_edges</a>.</li> </ul>
states	The network states among which pairwise transitions are to be counted. This should be a matrix (where the rows represent the binary form of the states) or a vector (for the binary form of a single state). The number of matrix columns (or the length of the vector) should match the number of network nodes.
steps	The number of time-steps, which should be a non-negative integer.
repeats	The number of repeats, which should be a positive integer.
asynchronous	If TRUE, the asynchronous update scheme is used, where a single node is updated at each time-step. In this case, update_prob indicates update probabilities. If FALSE, the synchronous update scheme is utilized.
update_prob	The probability of updating each variable (node) in each time-step when asynchronous = TRUE. If NULL (default), nodes are updated randomly based on a uniform distribution. If asynchronous = FALSE, this argument is ignored.

**Details**

Counting the number of transitions between each pair of states reveals the reachability of one state from another. This function performs simulations by starting from each state in states for steps time-steps and repeats iterations, and counts the number of transitions to other states in states.

**Value**

A matrix where each element (i, j) represents the number of transitions from the *i*th state to the *j*th state across steps time-steps and repeats iterations.





```
p_on <- runif(nrow(edges))
p_off <- runif(nrow(edges))
params <- list(p_on = p_on, p_off = p_off)

# Obtain frequency of pairwise transitions
pairwise_trans <- count_pairwise_trans(lac_operon_net, states = states,
  method = "PEW", params = params, steps = 100, repeats = 10)
```

---

**extract\_edges***Extract edges from a Boolean network*

---

### Description

Extracts the list of directed edges (links) from a given Boolean network.

### Usage

```
extract_edges(net, node_names = TRUE)
```

### Arguments

<code>net</code>	A network structure of the class <code>BooleanNetwork</code> from the <code>BoolNet</code> package.
<code>node_names</code>	If <code>TRUE</code> (default), the edges are returned by node names; otherwise, the edges are returned by node indices.

### Details

Since Boolean networks have a directed graph topology, this function extracts the list of directed edges from a given Boolean network.

### Value

A data frame where each row corresponds to a directed edge of the network and the two columns indicate the source and destination of each edge.

### Examples

```
# Load the example network
data(lac_operon_net)

# Extract edges from the network
edges <- extract_edges(lac_operon_net)
```

---

get\_reached\_states      *Obtain the reached states*

---

### Description

Obtains the reached states after simulating a Boolean network for a specified number of time-steps.

### Usage

```
get_reached_states(net, method = c("BNp", "SDDS", "PEW"), params, steps,
                  repeats = NULL, initial_states = NULL, asynchronous = TRUE,
                  update_prob = NULL)
```

### Arguments

net	A network structure of the class BooleanNetwork from the BoolNet package.
method	The parameterization method to be used. Options are: <ul style="list-style-type: none"> <li>• "BNp": Boolean network with perturbations.</li> <li>• "SDDS": Stochastic discrete dynamical systems.</li> <li>• "PEW": Boolean network with probabilistic edge weights.</li> </ul> Each method requires a different format for the params argument.
params	The parameter values depending on method: <ul style="list-style-type: none"> <li>• For method = "BNp", a single vector of probabilities, equal in length to the number of network nodes.</li> <li>• For method = "SDDS", a list of four equal-length vectors of probabilities: p00, p01, p10, and p11, each equal in length to the number of network nodes.</li> <li>• For method = "PEW", a list of two equal-length vectors of probabilities: p_on and p_off, each as long as the number of network edges, ordered according to <a href="#">extract_edges</a>.</li> </ul>
steps	The number of time-steps (non-negative integer) to simulate the network.
repeats	The number of repeats (positive integer). If two or more initial states are provided via initial_states, this argument is ignored. If NULL (default), then initial_states should not be NULL.
initial_states	The set of initial states as a matrix (where each row corresponds to the binary form of a state) or a vector (for the binary form of a single initial state). The number of matrix columns (or the length of the vector) should match the number of network nodes. The order of the nodes in the columns (or vector) is considered the same as net\$genes. If NULL (default), initial states are chosen randomly for repeats number of times based on a uniform distribution, requiring repeats not to be NULL.
asynchronous	If TRUE, the asynchronous update scheme is used, where a single node is updated at each time-step. In this case, update_prob indicates update probabilities. If FALSE, the synchronous update scheme is utilized.





```
# Multiple initial states are provided
reached_states <- get_reached_states(lac_operon_net, method = "PEW", params = params,
  steps = 100, initial_states = initial_states)
```

---

lac_operon_net	<i>The lactose operon Boolean network</i>
----------------	---

---

## Description

The *lactose* operon (*lac* operon) Boolean network as proposed by Veliz-Cuba and Stigler.

## Usage

```
data(lac_operon_net)
```

## Details

The data consists of an object `lac_operon_net` of the `BooleanNetwork` class from the `BoolNet` package, describing the *lac* operon gene regulatory network with 10 genes and 3 inputs. The three inputs collectively indicate the concentration of glucose and lactose. Based on the synchronous update scheme, when extracellular glucose is available, the *lac* operon is OFF (having one steady-state attractor where all genes are OFF). Otherwise, depending on the extracellular lactose concentration, the operon will be OFF, bistable (having two attractors), or ON (all genes are ON).

## References

Veliz-Cuba, A., & Stigler, B. (2011). Boolean models can explain bistability in the *lac* operon. *Journal of computational biology*, 18(6), 783-794.

## Examples

```
# load the network
data(lac_operon_net)

# the network is stored in a variable called 'lac_operon_net'
print(lac_operon_net)
```

# Index

[calc\\_convergence\\_time](#), [2](#), [3](#)  
[calc\\_node\\_activities](#), [2](#), [4](#), [5](#)  
[count\\_pairwise\\_trans](#), [3](#), [7](#)  
  
[extract\\_edges](#), [3](#), [5](#), [8](#), [10](#), [11](#)  
  
[get\\_reached\\_states](#), [3](#), [11](#)  
  
[lac\\_operon\\_net](#), [14](#)  
  
[pastboon \(pastboon-package\)](#), [2](#)  
[pastboon-package](#), [2](#)