Package 'pastboon'

August 22, 2024

Type Package

Title Simulation of Parameterized Stochastic Boolean Networks

Version 0.1.2

Description

A Boolean network is a particular kind of discrete dynamical system where the variables are simple binary switches. Despite its simplicity, Boolean network modeling has been a successful method to describe the behavioral pattern of various phenomena. 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: Boolean network with perturbations (BNp), described by Trairatphisan et al. (2013) <doi:10.1186/1478-811X-11-46>, stochastic discrete dynamical systems (SDDS), proposed by Murrugarra et al. (2012) <doi:10.1186/1687-4153-2012-5>, and Boolean network with probabilistic edge weights (PEW), presented by Deritei et al. (2022) <doi: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>

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 2024-08-22 08:50:03 UTC

Contents

`pastboon-package` *Simulation of Parameterized Stochastic Boolean Networks*

Description

A Boolean network is a particular kind of discrete dynamical system where the variables are simple binary switches. Despite its simplicity, Boolean network modeling has been a successful method to describe the behavioral pattern of various phenomena. 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 activities (the average state of the nodes at each time-step) in the form of a time-series can be calculated using `calc_node_activities`. By having a time-series representing node activities, 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 a node activity time-series 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 (i.e., time-series), where rows represent time-steps and columns represent nodes. It is the output of calc_node_activities.
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: • `"BNp"`: Boolean network with perturbations. • `"SDDS"`: Stochastic discrete dynamical systems. • `"PEW"`: Boolean network with probabilistic edge weights. Each method requires a different format for the `params` argument.
`params`	The parameter values depending on `method`: • For `method = "BNp"`, a single vector of probabilities, equal in length to the number of network nodes. • 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. • 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 `extract_edges`.
`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). It should be a vector of probabilities for each of the nodes which doesn't necessarily sum up to one. 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 node activity rates for all time-steps in the form of a time-series are 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. It should be a vector of probabilities for each of the nodes which sums up to one. If NULL (default), nodes are updated randomly based on a uniform distribution. If asynchronous = FALSE, this argument is ignored.

Details

By incorporating stochasticity into the update rule of a Boolean network and repeating the simulation several times, the average value of each node across the repeats can be considered as a continuous variable. This approach transforms discrete binary variables into continuous ones, enabling continuous analysis methods applicable for studying the dynamic behavior of the Boolean network. This function calculates the average value (i.e., node activity rate) of each network node at each time-step.

Value

If last_step = TRUE, a vector with a length equal to the number of network nodes, representing the activity rate of each node at the last time-step, is returned. If last_step = FALSE, a matrix with steps + 1 rows (where the first row corresponds to time-step 0) and length(net$genes) columns (representing node activities at each time-step) is returned. The order of the nodes in the vector or columns (depending on last_step) is the same as net$genes.

References

Golinelli, O., & Derrida, B. (1989). Barrier heights in the Kauffman model. *Journal De Physique*, 50(13), 1587-1601. Shmulevich, I., Dougherty, E. R., & Zhang, W. (2002). Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10), 1319-1331.

Shmulevich, I., Dougherty, E. R., & Zhang, W. (2002). Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10), 1319-1331.

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.

Examples

```
# >>>>>>>>>>>>>>>>  Load network and generate random initial states  <<<<<<<<<<<<<<<<

# Load the example network
data(lac_operon_net)

# Define plot function
plot_node_activities <- function(node_activities) {
  old_par <- par(no.readonly = TRUE)
  layout(matrix(c(1, 2), nrow = 1), width = c(4, 1))
```

```
  par(mar = c(5, 4, 4, 0))
  matplot(1:nrow(node_activities), node_activities, type = "l", frame = TRUE, lwd = 2,
      lty = 1, xlab = "Time-step", ylab = "Node activity")
  par(mar = c(5, 0, 4, 2))
  plot(c(0, 1), type = "n", axes = FALSE, xlab = "")
  legend("center", colnames(node_activities), col = seq_len(ncol(node_activities)),
      cex = 0.5, fill = seq_len(ncol(node_activities)))
  layout(1)
  par(old_par)
}

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: BNp  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Define the parameters for the BNp method
params <- rep(0.05, length(lac_operon_net$genes))

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

# Plot node activities
plot_node_activities(node_act)

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: SDDS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Define the 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)

# Plot node activities
plot_node_activities(node_act)

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: PEW  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

# Define the parameters for the PEW method
p_on <- runif(nrow(edges))
p_off <- runif(nrow(edges))
params <- list(p_on = p_on, p_off = p_off)

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

# Plot node activities
plot_node_activities(node_act)
```

`count_pairwise_trans` *Count pairwise transitions between a given set of states*

Description

Counts the frequencies of transitions between each pair of states from a given set of states.

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: • `"BNp"`: Boolean network with perturbations. • `"SDDS"`: Stochastic discrete dynamical systems. • `"PEW"`: Boolean network with probabilistic edge weights. Each method requires a different format for the `params` argument.
`params`	The parameter values depending on `method`: • For `method = "BNp"`, a single vector of probabilities, equal in length to the number of network nodes. • 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. • 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 `extract_edges`.
`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`. It should be a vector of probabilities for each of the nodes which sums up to one. 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.

References

Golinelli, O., & Derrida, B. (1989). Barrier heights in the Kauffman model. *Journal De Physique*, 50(13), 1587-1601.

Shmulevich, I., Dougherty, E. R., & Zhang, W. (2002). Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10), 1319-1331.

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.

Examples

```
# >>>>>>>>>>>>>>>>  Load network and generate random initial states  <<<<<<<<<<<<<<<<<<

# Load the example network
data(lac_operon_net)

# Generate some random states
states <- matrix(sample(c(0, 1), 10 * length(lac_operon_net$genes),
    replace = TRUE), nrow = 10, ncol = length(lac_operon_net$genes))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: BNp  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Define the parameters for the BNp method
params <- rep(0.05, length(lac_operon_net$genes))

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

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: SDDS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Define the parameters for the SDDS method
props <- rep(0.95, length(lac_operon_net$genes))
```

```
params <- list(p00 = props, p01 = props, p10 = props, p11 = props)

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

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: PEW  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

# Define the parameters for the PEW method
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

`net`	A network structure of the class `BooleanNetwork` from the `BoolNet` package.
`node_names`	If TRUE (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: • `"BNp"`: Boolean network with perturbations. • `"SDDS"`: Stochastic discrete dynamical systems. • `"PEW"`: Boolean network with probabilistic edge weights. Each method requires a different format for the `params` argument.
`params`	The parameter values depending on `method`: • For `method = "BNp"`, a single vector of probabilities, equal in length to the number of network nodes. • 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. • 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 `extract_edges`.
`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.

`update_prob` The probability of updating each variable (node) in each time-step when `asynchronous = TRUE`. It should be a vector of probabilities for each of the nodes which sums up to one. If `NULL` (default), nodes are updated randomly based on a uniform distribution. If `asynchronous = FALSE`, this argument is ignored.

Details

This function returns the reached states (the states in the last time-step) after simulating a network for `steps` time-steps and repeating it for `repeats` number of times. If `initial_states` is `NULL`, then the initial states are chosen randomly based on a uniform distribution for `repeats` number of times, resulting in `repeats` number of reached states. If two or more initial states are provided by the user, then the `repeats` argument is ignored, and one reached state is returned for each initial state. If `repeats` is `NULL`, the number of returned reached states equals the number of initial states (one reached state for each initial state). The arguments `repeats` and `initial_states` should not both be `NULL` simultaneously.

Value

A matrix where each row is the binary form of a reached state, and each column corresponds to a network node. The order of the nodes in the columns is the same as `net$genes`.

References

Golinelli, O., & Derrida, B. (1989). Barrier heights in the Kauffman model. *Journal De Physique*, 50(13), 1587-1601.

Shmulevich, I., Dougherty, E. R., & Zhang, W. (2002). Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10), 1319-1331.

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.

Examples

```
# >>>>>>>>>>>>>>>>  Load network and generate random initial states  <<<<<<<<<<<<<<<<

# Load the example network
data(lac_operon_net)

# Generate a single initial state
initial_state <- sample(c(0, 1), length(lac_operon_net$genes), replace = TRUE)

# Generate multiple (10) initial states
initial_states <- matrix(sample(c(0, 1), 10 * length(lac_operon_net$genes),
    replace = TRUE), nrow = 10, ncol = length(lac_operon_net$genes))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: BNp  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Define the parameters for the BNp method
params <- rep(0.05, length(lac_operon_net$genes))

# No initial states are provided
reached_states <- get_reached_states(lac_operon_net, method = "BNp", params = params,
    steps = 100, repeats = 10)

# A single initial state is provided
reached_states <- get_reached_states(lac_operon_net, method = "BNp", params = params,
    steps = 100, initial_states = initial_state, repeats = 10)

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

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: SDDS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

# No initial states are provided
reached_states <- get_reached_states(lac_operon_net, method = "SDDS", params = params,
    steps = 100, repeats = 10)

# A single initial state is provided
reached_states <- get_reached_states(lac_operon_net, method = "SDDS", params = params,
    steps = 100, initial_states = initial_state, repeats = 10)

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

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Method: PEW  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

```
# Define the parameters for the PEW method
p_on <- runif(nrow(edges))
p_off <- runif(nrow(edges))
params <- list(p_on = p_on, p_off = p_off)

# No initial states are provided
reached_states <- get_reached_states(lac_operon_net, method = "PEW", params = params,
    steps = 100, repeats = 10)

# A single initial state is provided
reached_states <- get_reached_states(lac_operon_net, method = "PEW", params = params,
    steps = 100, initial_states = initial_state, repeats = 10)

# 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` *The lactose operon Boolean network*

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 class `BooleanNetwork` (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