Package ‘wdnet’

September 6, 2023

Title  Weighted and Directed Networks
Version  1.2.2
Date  2023-09-05
Maintainer  Yelie Yuan <yelie.yuan@uconn.edu>

Description  Implementations of network analysis including
(1) assortativity coefficient of weighted and directed networks,
Yuan, Yan and Zhang (2021) <doi:10.1093/comnet/cnab017>,
(2) centrality measures for weighted and directed networks,
Opsahl, Agneessens and Skvoretz (2010) <doi:10.1016/j.socnet.2010.03.006>,
(3) clustering coefficient of weighted and directed networks,
Fagiolo (2007) <doi:10.1103/PhysRevE.76.026107> and
(4) rewiring networks with given assortativity coefficients,
Wang, Yan, Yuan and Zhang (2022) <doi:10.1007/s11222-022-10161-8>,
(5) preferential attachment network generation,

Depends  R (>= 4.1.0)
License  GPL (>= 3.0)
Encoding  UTF-8
Imports  CVXR, igraph, Matrix, rARPACK, RcppXPtrUtils, stats, wdm
LinkingTo  Rcpp, RcppArmadillo

BugReports  https://gitlab.com/wdnetwork/wdnet/-/issues
URL  https://gitlab.com/wdnetwork/wdnet
RoxygenNote  7.2.3
Suggests  testthat (>= 3.0.0)
NeedsCompilation  yes

Author  Yelie Yuan [aut, cre],
        Tiandong Wang [aut],
        Jun Yan [aut],
        Panpan Zhang [aut]
Description

`+` is used to combine components to control the PA network generation process. Available components are `rpa_control_scenario()`, `rpa_control_edgeweight()`, `rpa_control_newedge()`, `rpa_control_preference()` and `rpa_control_reciprocal()`.

Usage

```r
## S3 method for class 'rpacontrol'
e1 + e2
```

Arguments

- `e1` A list of class `rpacontrol`.
- `e2` A list of class `rpacontrol`. 
Value

A list of class rpacontrol with components from e1 and e2.

Examples

```r
control <- rpa_control_scenario(alpha = 0.5, beta = 0.5) +
  rpa_control_preference(
    ftype = "customized",
    spref = "pow(outs, 2) + 1",
    tpref = "pow(ins, 2) + 1"
  )

control <- rpa_control_scenario(alpha = 1) +
  rpa_control_edgeweight(
    sampler = function(n) rgamma(n, shape = 5, scale = 0.2)
  )
```

adj_to_wdnet

Creates a wdnet object using an adjacency matrix

Description

Creates a wdnet object using an adjacency matrix

Usage

```r
adj_to_wdnet(adj, directed = TRUE, weighted = TRUE, nodegroup, ...)
```

Arguments

- `adj` An adjacency matrix used to extract edgelist and edgeweight using igraph.
- `directed` Logical, whether the network is directed (TRUE) or undirected (FALSE). If `adj` is asymmetric, the network is directed.
- `weighted` Logical, whether the network is weighted (TRUE) or unweighted (FALSE).
- `nodegroup` A numeric vector of node groups.
- `...` Additional components to be added to the wdnet object.

Value

A wdnet object with the specified `adj`.

Examples

```r
adj <- matrix(c(0, 1, 2, 0), nrow = 2, ncol = 2, byrow = TRUE)
adj_to_wdnet(adj = adj, directed = TRUE, weighted = FALSE)
```
assortcoef  
Compute the assortativity coefficient(s) for a network.

Description
Compute the assortativity coefficient(s) for a network.

Usage
assortcoef(netwk, edgelist, edgeweight, adj, directed, f1, f2)

Arguments
- netwk: A `wdnet` object that represents the network. If `NULL`, the function will compute the coefficient using either `edgelist` and `edgeweight`, or `adj`.
- edgelist: A two-column matrix representing edges.
- edgeweight: A numeric vector of edge weights with the same length as the number of rows in `edgelist`. If `NULL`, all edges will be assigned weight 1.
- adj: The adjacency matrix of a network.
- directed: Logical. Indicates whether the edges in `edgelist` or `adj` are directed. It will be omitted if `netwk` is provided.
- f1: A vector representing the first feature of existing nodes. The number of nodes should be equal to the length of both `f1` and `f2`. Defined for directed networks. If `NULL`, out-strength will be used.
- f2: A vector representing the second feature of existing nodes. Defined for directed networks. If `NULL`, in-strength will be used.

Value
Assortativity coefficient for undirected networks, or a list of four assortativity coefficients for directed networks.

Note
When the adjacency matrix is binary (i.e., directed but unweighted networks), `assortcoef` returns the assortativity coefficient proposed in Foster et al. (2010).

References
Examples

```r
set.seed(123)
control <- rpa_control_edgeweight(
  sampler = function(n) rgamma(n, shape = 5, scale = 0.2)
)
netwk <- rpanet(nstep = 10^4, control = control)
ret <- assortcoef(netwk)
ret <- assortcoef(
  edgelist = netwk$edgelist,
  edgeweight = netwk$edge.attr$weight,
  directed = TRUE
)
```

---

### centrality

#### Centrality measures

**Description**

Computes the centrality measures of the nodes in a weighted and directed network.

**Usage**

```r
centrality(
  netwk,
  adj,
  edgelist,
  edgeweight,
  directed = TRUE,
  measure = c("degree", "closeness", "wpr"),
  degree.control = list(alpha = 1, mode = "out"),
  closeness.control = list(alpha = 1, mode = "out", method = "harmonic", distance = FALSE),
  wpr.control = list(gamma = 0.85, theta = 1, prior.info = NULL)
)
```

**Arguments**

- `netwk`: A `wdnet` object that represents the network. If `NULL`, the function will compute the coefficient using either `edgelist` and `edgeweight`, or `adj`.
- `adj`: An adjacency matrix of a weighted and directed network.
- `edgelist`: A two-column matrix representing edges of a directed network.
- `edgeweight`: A vector representing the weight of edges.
- `directed`: Logical. Indicates whether the edges in `edgelist` or `adj` are directed.
- `measure`: Which measure to use: "degree" (degree-based centrality), "closeness" (closeness centrality), or "wpr" (weighted PageRank centrality)?
degree.control  A list of parameters passed to the degree centrality measure:

- alpha A tuning parameter. The value of alpha must be nonnegative. By
  convention, alpha takes a value from 0 to 1 (default).
- mode Which mode to compute: "out" (default) or "in"? For undirected
  networks, this setting is irrelevant.

closeness.control

A list of parameters passed to the closeness centrality measure:

- alpha A tuning parameter. The value of alpha must be nonnegative. By
  convention, alpha takes a value from 0 to 1 (default).
- mode Which mode to compute: "out" (default) or "in"? For undirected
  networks, this setting is irrelevant.
- method Which method to use: "harmonic" (default) or "standard"?
- distance Whether to consider the entries in the adjacency matrix as dis-
  tances or strong connections. The default setting is FALSE.

wpr.control  A list of parameters passed to the weighted PageRank centrality measure:

- gamma The damping factor; it takes 0.85 (default) if not given.
- theta A tuning parameter leveraging node degree and strength; theta =
  0 does not consider edge weight; theta = 1 (default) fully considers edge
  weight.
- prior.info Vertex-specific prior information for restarting when arriving at a
  sink. When it is not given (NULL), a random restart is implemented.

Value

A list of node names and associated centrality measures

Note

The degree-based centrality measure is an extension of function strength in package igraph and
and alternative of function degree_w in package tnet.

The closeness centrality measure is an extension of function closeness in package igraph and
function closeness_w in package tnet. The method of computing distances between vertices is the
Dijkstra’s algorithm.

The weighted PageRank centrality measure is an extension of function page_rank in package igraph.

References

  ematik, 1, 269–271.
  167–256.
degree and shortest paths. Social Networks, 32, 245–251.

Zhang, P., Zhao, J. and Yan, J. (2020+) Centrality measures of networks with application to world input-output tables

**Examples**

```r
## Generate a network according to the Erdos-Renyi model of order 20
## and parameter p = 0.3
edge_ER <- rbinom(400, 1, 0.3)
weight_ER <- sapply(edge_ER, function(x) x * sample(3, 1))
adj_ER <- matrix(weight_ER, 20, 20)

mydegree <- centrality(
  adj = adj_ER,
  measure = "degree", degree.control =
  list(alpha = 0.8, mode = "in")
)

myclose <- centrality(  
  adj = adj_ER,
  measure = "closeness", closeness.control =
  list(alpha = 0.8, mode = "out", method = "harmonic", distance = FALSE)
)

mywpr <- centrality(  
  adj = adj_ER,
  measure = "wpr", wpr.control =
  list(gamma = 0.85, theta = 0.75)
)
```

---

**clustcoef**

*Directed clustering coefficient*

**Description**

Compute the clustering coefficient of a weighted and directed network.

**Usage**

```r
clustcoef(
  netwk, edgelist, edgeweight, adj, directed = TRUE, method = c("Clemente", "Fagiolo"), isolates = 0
)
```
Arguments

netwk A `wdnet` object that represents the network. If `NULL`, the function will compute the coefficient using either `edgelist`, `edgeweight`, or `adj`.

edgelist A two-column matrix, each row represents a directed edge of the network.

edgeweight A vector representing the weight of edges.

adj An adjacency matrix of a weighted and directed network.

directed Logical. Indicates whether the edges in `edgelist` or `adj` are directed.


isolates Binary, defines how to treat vertices with degree zero and one. If 0, then their clustering coefficient is returned as 0 and are included in the averaging. Otherwise, their clustering coefficient is NaN and are excluded in the averaging. Default value is 0.

Value

Lists of local clustering coefficients (in terms of a vector), global clustering coefficient (in terms of a scalar) and number of weighted directed triangles (in terms of a vector) based on `total`, `in`, `out`, `middleman` (`middle`), or `cycle` triplets.

Note

Self-loops (if exist) are removed prior to the computation of clustering coefficient. When the adjacency matrix is symmetric (i.e., undirected but possibly unweighted networks), `clustcoef` returns local and global clustering coefficients proposed by Barrat et al. (2010).

References


Examples

```r
## Generate a network according to the Erdős-Rényi model of order 20
## and parameter p = 0.3
edge_ER <- rbinom(400, 1, 0.3)
weight_ER <- sapply(edge_ER, function(x) x * sample(3, 1))
adj_ER <- matrix(weight_ER, 20, 20)
mycc <- clustcoef(adj = adj_ER, method = "Clemente")
system.time(mycc)
```
cvxr_control

Parameters passed to CVXR::solve.

Description
Defined for the convex optimization problems for solving eta.

Usage
cvxr_control(
  solver = "ECOS",
  ignore_dcp = FALSE,
  warm_start = FALSE,
  verbose = FALSE,
  parallel = FALSE,
  gp = FALSE,
  feastol = 1e-05,
  reltol = 1e-05,
  abstol = 1e-05,
  num_iter = NULL,
  ...
)

Arguments

  solver  (Optional) A string indicating the solver to use. Defaults to "ECOS".
  ignore_dcp (Optional) A logical value indicating whether to override the DCP check for a problem.
  warm_start (Optional) A logical value indicating whether the previous solver result should be used to warm start.
  verbose (Optional) A logical value indicating whether to print additional solver output.
  parallel (Optional) A logical value indicating whether to solve in parallel if the problem is separable.
  gp (Optional) A logical value indicating whether the problem is a geometric program. Defaults to FALSE.
  feastol The feasible tolerance on the primal and dual residual. Defaults to 1e-5.
  reltol The relative tolerance on the duality gap. Defaults to 1e-5.
  abstol The absolute tolerance on the duality gap. Defaults to 1e-5.
  num_iter The maximum number of iterations.
  ... Additional options that will be passed to the specific solver. In general, these options will override any default settings imposed by CVXR.

Value
A list containing the parameters.
Examples

control <- cvxr_control(solver = "OSQP", abstol = 1e-5)


dprewire

Degree preserving rewiring.

Description

Rewire a given network to have predetermined assortativity coefficient(s) while preserving node degree.

Usage

dprewire(
  netwk, edgelist, directed, adj,
  target.assortcoef = list(outout = NULL, outin = NULL, inout = NULL, inin = NULL),
  control = list(iteration = 200, nattempts = NULL, history = FALSE, cvxr_control = cvxr_control(), eta.obj = function(x) 0),
  eta
)

Arguments

  netwk     A wdnet object representing an unweighted network. If NULL, the function will construct a network using either edgelist, or adj.
  edgelist  A two column matrix, each row represents an edge of the network.
  directed  Logical, whether the network is directed or not. It will be ignored if netwk is provided.
  adj       An adjacency matrix of an unweighted network.
  target.assortcoef
    For directed networks, it is a list represents the predetermined value or range of assortativity coefficients. For undirected networks, it is a constant between -1 to 1. It will be ignored if eta is provided.
  control  A list of parameters for controlling the rewiring process and the process for solving eta.
    • iteration An integer, represents the number of rewiring iterations. Each iteration consists of nattempts rewiring attempts. The assortativity coefficient(s) of the network will be recorded after each iteration.
    • nattempts An integer representing the number of rewiring attempts for each iteration. Default value equals the number of rows of edgelist.
    • history Logical, whether the rewiring attempts should be recorded and returned.
dpewire

- **eta.obj** A convex function of \( \eta \) to be minimized when solving for \( \eta \) with given `target.assortcoef`. Defaults to 0. It will be ignored if \( \eta \) is provided.
- **cvxr_control** A list of parameters passed to `CVXR::solve()` for solving \( \eta \) with given `target.assortcoef`. It will be ignored if \( \eta \) is provided.

**eta** A matrix represents the target network structure. If specified, `target.assortcoef` will be ignored. For directed networks, the element at row “i-j” and column “k-l” represents the proportion of directed edges linking a source node with out-degree i and in-degree j to a target node with out-degree k and in-degree l. For undirected networks, \( \eta \) is symmetric, the summation of the elements at row ”i”, column ”j” and row ”j”, column ”i” represents the proportion of edges linking to a node with degree i and a node with degree j.

**Details**

The algorithm first solves for an appropriate \( \eta \) using `target.assortcoef`, `eta.obj`, and `cvxr_control`, then proceeds to the rewiring process and rewire the network towards the solved \( \eta \). If \( \eta \) is given, the algorithm will skip the first step. This function only works for unweighted networks.

Each rewiring attempt samples two rows from `edgelist`, for instance Edge 1: \((v_1, v_2)\) and Edge 2: \((v_3, v_4)\). For directed networks, if the rewiring attempt is accepted, the sampled edges are rewired as \((v_1, v_4), (v_3, v_2)\); for undirected networks, the algorithm try to rewire the sampled edges as \{\(v_1, v_4\), \(v_3, v_2\)\} (type 1) or \{\(v_1, v_3\), \(v_2, v_4\)\} (type 2), each with probability 1/2.

**Value**

Rewired network; assortativity coefficient(s) after each iteration; rewiring history (including the index of sampled edges and rewiring result) and solver results.

**Examples**

```r
set.seed(123)
netwk1 <- rpanet(1e4, control = rpa_control_scenario(
  alpha = 0.4, beta = 0.3, gamma = 0.3
))
## rewire a directed network
target.assortcoef <- list("outout" = -0.2, "outin" = 0.2)
ret1 <- dpewire(
  netwk = netwk1,
  target.assortcoef = target.assortcoef,
  control = list(iteration = 200)
)
plot(ret1$assortcoef$Iteration, ret1$assortcoef"outout")
plot(ret1$assortcoef$Iteration, ret1$assortcoef"outin")

## rewire an undirected network
netwk2 <- rpanet(1e4,
  control = rpa_control_scenario(
    alpha = 0.3, beta = 0.1, gamma = 0.3, xi = 0.3
)```
```
),
  initial.network = list(  
    directed = FALSE  
  )
)  
ret2 <- dprewire(  
  netwk = netwk2,
  target.assortcoef = 0.3,
  control = list(  
    iteration = 300, eta.obj = CVXR::norm2,
    history = TRUE
  )
)  
plot(ret2$assortcoef$Iteration, ret2$assortcoef$Value)
```

dprewire.range  

Range of assortativity coefficients.

**Description**

The assortativity coefficient of a given network may not reach all the values between -1 and 1 via degree preserving rewiring. This function calculates the range of assortativity coefficients achievable through degree preserving rewiring. The algorithm is designed for unweighted networks.

**Usage**

```
dprewire.range(  
  netwk,
  edgelist,
  adj,
  directed,
  which.range = c("outout", "outin", "inout", "inin"),
  control = cvxr_control(),
  target.assortcoef = list(outout = NULL, outin = NULL, inout = NULL, inin = NULL)
)
```

**Arguments**

- `netwk` A wdnet object representing an unweighted network. If NULL, the function will construct a network using either edgelist or adj.
- `edgelist` A two-column matrix, where each row represents an edge of the network.
- `adj` An adjacency matrix of an unweighted network.
- `directed` Logical, whether the network is directed or not. It will be ignored if `netwk` is provided.
- `which.range` The type of interested assortativity coefficient. For directed networks, it takes one of the values: "outout", "outin", "inout" and "inin". It will be ignored if the network is undirected.
control A list of parameters passed to CVXR::solve() for solving an appropriate eta, given the constraints target.assortcoef.

target.assortcoef A list of constraints, it contains the predetermined value or range imposed on assortativity coefficients other than which.range. It will be ignored if the network is undirected.

Details

The ranges are computed using convex optimization. The optimization problems are defined and solved via the R package CVXR. For undirected networks, the function returns the range of the assortativity coefficient. For directed networks, the function computes the range of which.range while other assortativity coefficients are restricted through target.assortcoef.

Value

Returns the range of the selected assortativity coefficient and the results from the solver.

Examples

```r
set.seed(123)
netwk <- rpanet(5e3,
    control =
        rpa_control_scenario(alpha = 0.5, beta = 0.5)
)
ret1 <- dprewire.range(
    netwk = netwk, which.range = "outin",
    target.assortcoef = list("outout" = c(-0.3, 0.3), "inout" = 0.1)
)
ret1$range
```

edgelist_to_wdnet

Creates a wdnet object using edgelist.

Description

Creates a wdnet object using edgelist.

Usage

```r
edgelist_to_wdnet(edgelist, edgweight, directed, nodegroup, ...)
```
igraph_to_wdnet

Converts an igraph object to a wdnet object

Description
Converts an igraph object to a wdnet object

Usage
igraph_to_wdnet(g)

Arguments
    g          An igraph object.

Value
A wdnet object.

Examples
    g <- igraph::sample_pa(50)
    netwk <- igraph_to_wdnet(g)
is_wdnet  Checks if the input is a wdnet object

Description
Checks if the input is a wdnet object

Usage
is_wdnet(netwk)

Arguments
netwk  A wdnet object.

Value
Logical, TRUE if argument netwk is a wdnet object.

Examples
netwk <- rpanet(nstep = 1e3)
is_wdnet(netwk)

plot.wdnet  Plots the input network

Description
Plots the input network via igraph::plot.igraph().

Usage
## S3 method for class 'wdnet'
plot(x, ...)

Arguments
x  A wdnet object.
...
Additional parameters passed to igraph::plot.igraph().

Value
Returns NULL, invisibly.
print.rpacontrol | Prints rpacontrol objects

Description

These functions print rpacontrol objects in the terminal. print.rpacontrol() shows only the current controls, whereas summary.rpacontrol() includes both specified controls and the unspecified controls that use default values.

Usage

```r
define S3 method for class 'rpacontrol'
  print(x, ...)
define S3 method for class 'rpacontrol'
  summary(object, ...)
```

Arguments

- `x`: An object of class rpacontrol.
- `...`: Additional arguments.
- `object`: An object of class rpacontrol.

Value

Returns the controls invisibly.

Examples

```r
control <- rpa_control_scenario()
print(control)
```

print.wdnet | Prints the input network

Description

These functions print a network to the terminal.
**rpacontrol**

### Usage

```r
## S3 method for class 'wdnet'
print(x, node.attrs = TRUE, edge.attrs = TRUE, max.lines = 5, ...)

## S3 method for class 'wdnet'
summary(object, ...)
```

### Arguments

- **x**: A `wdnet` object.
- **node.attrs**: Logical, whether to print node attributes, if available.
- **edge.attrs**: Logical, whether to print edge attributes, if available.
- **max.lines**: Integer, the maximum number of lines of edgelist and node attributes to print. The rest of the output will be truncated.
- **...**: Additional arguments.
- **object**: The graph of which the summary will be printed.

### Details

`summary.wdnet` prints the number of nodes and edges, preference functions, and whether the network is directed, weighted. `print.wdnet` prints the same information, and also lists some edges and node attributes, if available. Edge scenarios are 0: from initial network; 1: alpha; 2: beta; 3: gamma; 4: xi; 5: rho; 6: reciprocal.

### Description

The `rpacontrol` object is designed to control the Preferential Attachment (PA) network generation process within the `rpanet()` function. It can have the following components:

- **scenario**: controls the edge scenarios at each step. For more information, please refer to `rpa_control_scenario()`.
- **edgeweight**: controls the weight of the edges; see `rpa_control_edgeweight()` for details.
- **newedge**: controls the creation of new edges at each step; see `rpa_control_newedge()` for details.
- **preference**: sets preference functions; see `rpa_control_preference()` for details.
- **reciprocal**: controls the creation of reciprocal edges; see `rpa_control_reciprocal()` for details.
rpanet Generate PA networks.

Description

Generate preferential attachment (PA) networks with linear or non-linear preference functions.

Usage

rpanet(
  nstep,
  initial.network = list(edgelist = matrix(c(1, 2), nrow = 1), edgeweight = 1, directed = TRUE),
  control,
  method = c("binary", "linear", "bagx", "bag")
)

Arguments

nstep Number of steps.
initial.network A wnnet object or a list representing the initial network. By default, initial.network has one directed edge from node 1 to node 2 with weight 1. It can contain the following components: a two-column matrix edgelist representing the edges; a vector edgeweight representing the weight of edges; a logical argument directed indicating whether the initial network is directed. If edgeweight is not specified, all edges from the initial network are assumed to have weight 1. In addition, an integer vector nodegroup can be added to the list for specifying node groups; nodegroup is defined for directed networks, if NULL, all nodes from the seed network are assumed to be in group 1.
control An rpacontrol object controlling the PA network generation process. If not specified, all the control parameters will be set to default. For more details, see rpa_control_scenario(), rpa_control_newedge(), rpa_control_edgeweight(), rpa_control_preference and rpa_control_reciprocal(). Under the default setup, at each step, a new edge of weight 1 is added from a new node A to an existing node B (alpha scenario), where B is chosen with probability proportional to its in-strength + 1.
method Which method to use: binary, linear, bagx or bag. For bag and bagx methods, beta.loop must be TRUE, default preference functions must be used, and sparams should be set to c(1, 1, 0, 0, a), tparams to c(0, 0, 1, 1, b), and param to c(1, c), where a, b, and c are non-negative constants; furthermore, reciprocal edges and sampling without replacement are not considered, i.e., option rpa_control_reciprocal() must be set as default, snode.replace, tnode.replace and node.replace must be TRUE. In addition, bag method only works for unweighted networks and does not consider multiple edges, i.e., rpa_control_edgeweight() and rpa_control_newedge() must be set as default.
Value

Returns a \texttt{wdnet} object that includes the following components:

- directed: Logical, whether the network is directed.
- weighted: Logical, whether the network is weighted.
- edgelist: A two-column matrix representing the edges.
- edge.attr: A data frame including edge weights and edge scenarios (0: from initial network; 1: \texttt{alpha}; 2: \texttt{beta}; 3: \texttt{gamma}; 4: \texttt{xi}; 5: \texttt{rho}; 6: reciprocal edge).
- node.attr: A data frame including node out- and in-strength, node source and target preference scores (for directed networks), node strength and preference scores (for undirected networks), and node group (if applicable).
- newedge: The number of new edges at each step, including reciprocal edges.
- control: An \texttt{rpacontrol} object that is used to generate the network.

Note

The binary method implements binary search algorithm; linear represents linear search algorithm; bag method implements the algorithm from Wan et al. (2017); bagx puts all the edges into a bag, then samples edges and find the source/target node of the sampled edge.

References


Examples

```r
# Control edge scenario and edge weight through \texttt{rpa_control_scenario()} # and \texttt{rpa_control_edgeweight()}, respectively,
# while keeping \texttt{rpa_control_newedge()},
# \texttt{rpa_control_preference()} and \texttt{rpa_control_reciprocal()} as default.
set.seed(123)
control <- rpa_control_scenario(alpha = 0.5, beta = 0.5) +
  rpa_control_edgeweight(
    sampler = function(n) rgamma(n, shape = 5, scale = 0.2)
  )
ret1 <- rpanet(nstep = 1e3, control = control)

# In addition, set node groups and probability of creating reciprocal edges.
control <- control + rpa_control_reciprocal(
  group.prob = c(0.4, 0.6),
  recip.prob = matrix(runif(4), ncol = 2)
)
ret2 <- rpanet(nstep = 1e3, control = control)

# Further, set the number of new edges in each step as Poisson(2) + 1 and use # \texttt{ret2} as a seed network.
control <- control + rpa_control_newedge(
  sampler = function(n) rpois(n, lambda = 2) + 1
```

rpa_control_newedge

) 
ret3 <- rpanet(nstep = 1e3, initial.network = ret2, control = control)

rpa_control_edgeweight

Control weight of new edges. Defined for rpanet.

Description

Control weight of new edges. Defined for rpanet.

Usage

rpa_control_edgeweight(sampler = NULL)

Arguments

sampler A function used for sampling edge weights. If NULL, all new edges will default to a weight of 1. If a function is provided, it must accept a single argument, n, and return a vector of length n that represents the sampled edge weights.

Value

A list of class rpacontrol containing the sampler function.

Examples

# Sample edge weights from Gamma(5, 0.2).
my_gamma <- function(n) rgamma(n, shape = 5, scale = 0.2)
control <- rpa_control_edgeweight(
  sampler = my_gamma
)

rpa_control_newedge

Control new edges in each step. Defined for rpanet.

Description

Control new edges in each step. Defined for rpanet.
Usage

rpa_control_newedge(
  sampler = NULL,
  snode.replace = TRUE,
  tnode.replace = TRUE,
  node.replace = TRUE
)

Arguments

sampler A function used for sampling the number of new edges to be added at each step. If NULL, one new edge will be added at each step. If a function is provided, it must accept a single argument, n, and return a vector of length n that represents the sampled number of new edges.

snode.replace Logical. Determines whether the source nodes in the same step should be sampled with replacement. Defined for directed networks.

tnode.replace Logical. Determines whether the target nodes in the same step should be sampled with replacement. Defined for directed networks.

node.replace Logical. Determines whether the nodes in the same step should be sampled with replacement. Defined for undirected networks. If FALSE, self-loops will not be allowed under beta scenario.

Value

A list of class rpacontrol with components sampler, snode.replace, tnode.replace and node.replace with meanings as explained under 'Arguments'.

Examples

my_rpois <- function(n) rpois(n, lambda = 2) + 1
control <- rpa_control_newedge(
  sampler = my_rpois,
  node.replace = FALSE
)

Description

Set preference function(s). Defined for rpanet.
rpa_control_preference

Usage

rpa_control_preference(
  ftype = c("default", "customized"),
  sparms = c(1, 1, 0, 0, 1),
  tparams = c(0, 0, 1, 1, 1),
  params = c(1, 1),
  spref = "outs + 1",
  tpref = "ins + 1",
  pref = "s + 1"
)

Arguments

ftype Preference function type. Either "default" or "customized". "customized" preference functions require "binary" or "linear" generation methods. If using default preference functions, sparms, tparams and params must be specified. If using customized preference functions, spref, tpref and pref must be specified.


params A numerical vector of length 2 giving the parameters of the default preference function. Defined for undirected networks. Probability of choosing an existing node is proportional to strength*params[1] + params[2].

spref Character expression or an object of class XPtr giving the customized source preference function. Defined for directed networks. Default value is "outs + 1", i.e., node out-strength + 1. See Details and Examples for more information.

tpref Character expression or an object of class XPtr giving the customized target preference function. Defined for directed networks. Default value is "ins + 1", i.e., node in-strength + 1.

pref Character expression or an object of class XPtr giving the customized preference function. Defined for undirected networks. Default value is "s + 1", i.e, node strength + 1.

Details

If choosing customized preference functions, spref, tpref and pref will be used and the network generation method must be "binary" or "linear". spref (tpref) defines the source (target) preference function, it can be a character expression or an object of class XPtr.

- Character expression: it must be a one-line C++ style expression of outs (node out-strength) and ins (node in-strength). For example, "pow(outs, 2) + 1", "pow(outs, 2) + pow(ins,
2) + 1", etc. The expression will be used to define an XPtr via `RcppXPtrUtils::cppXPtr`. The XPtr will be passed to the network generation function. The expression must not have variables other than `outs` and `ins`.

- **XPtr**: an external pointer wrapped in an object of class `XPtr` defined via `RcppXPtrUtils::cppXPtr`. An example for defining an XPtr with C++ source code is included in Examples. For more information about passing function pointers, see [https://gallery.rcpp.org/articles/passing-cpp-function-pointers-rcppxptrutils/](https://gallery.rcpp.org/articles/passing-cpp-function-pointers-rcppxptrutils/). Please note the supplied C++ function accepts two double arguments and returns a double. The first and second arguments represent node out- and in-strength, respectively. Note that the XPtr will be invalid and cannot be used to control network generation in another separate R session. Therefore, we recommend preserving the source code of your preference function for future use.

`pref` is defined analogously. If using character expression, it must be a one-line C++ style expression of `s` (node strength). If using XPtr, the supplied C++ function accepts only one double argument and returns a double.

**Value**

A list of class `rpacontrol` with components `ftype`, `sparams`, `tparams`, `params` or `ftype`, `spref`, `tpref`, `pref` with function pointers `spref.pointer`, `tpref.pointer`, `pref.pointer`.

**Examples**

```r
# Set source preference as out-strength^2 + in-strength + 1,
# target preference as out-strength + in-strength^2 + 1.
# 1. use default preference functions
ctrl1 <- rpa_control_preference(
  ftype = "default",
  sparams = c(1, 2, 1, 1, 1), tparams = c(1, 1, 2, 1)
)
# 2. use character expressions
ctrl2 <- rpa_control_preference(
  ftype = "customized",
  spref = "pow(outs, 2) + ins + 1", tpref = "outs + pow(ins, 2) + 1"
)
# 3. define XPtr's with C++ source code
spref.pointer <- RcppXPtrUtils::cppXPtr(
  code = 
  "double spref(double outs, double ins) {return pow(outs, 2) + ins + 1;};"
)
tpref.pointer <- RcppXPtrUtils::cppXPtr(
  code = 
  "double tpref(double outs, double ins) {return outs + pow(ins, 2) + 1;};"
)
ctrl3 <- rpa_control_preference(
  ftype = "customized",
  spref = spref.pointer,
  tpref = tpref.pointer
)
ret <- rpanet(1e5, control = ctrl3)
```
Control reciprocal edges. Defined for rpanet.

**Usage**

```r
rpa_control_reciprocal(
  group.prob = NULL,
  recip.prob = NULL,
  selfloop.recip = FALSE
)
```

**Arguments**

- `group.prob` A vector of probability weights for sampling the group of new nodes. Defined for directed networks. Groups are from 1 to `length(group.prob)`. Its length must equal to the number of rows of `recip.prob`.

- `recip.prob` A square matrix giving the probability of adding a reciprocal edge after a new edge is introduced. Defined for directed networks. Its element \( p_{ij} \) represents the probability of adding a reciprocal edge from node \( A \) in group \( i \) to node \( B \) in group \( j \) immediately after a directed edge from \( B \) to \( A \) is added.

- `selfloop.recip` Logical, whether reciprocal edge of self-loops are allowed.

**Value**

A list of class `rpacontrol` with components `group.prob`, `recip.prob`, and `selfloop.recip` with meanings as explained under 'Arguments'.

**Examples**

```r
control <- rpa_control_reciprocal(
  group.prob = c(0.4, 0.6),
  recip.prob = matrix(runif(4), ncol = 2)
)
```
Control edge scenarios. Defined for rpanet.

Usage

rpa_control_scenario(
  alpha = 1,
  beta = 0,
  gamma = 0,
  xi = 0,
  rho = 0,
  beta.loop = TRUE,
  source.first = TRUE
)

Arguments

alpha  Probability of adding an edge from a new node to an existing node.
beta   Probability of adding an edge between existing nodes.
gamma  Probability of adding an edge from an existing node to a new node.
xi     Probability of adding an edge between two new nodes.
rho    Probability of adding a new node with a self-loop.
beta.loopLogical. Determines whether self-loops are allowed under the beta scenario. Default value is TRUE.
source.firstLogical. Defined for beta scenario edges of directed networks. If TRUE, the source node of a new edge is sampled from existing nodes before the target node is sampled; if FALSE, the target node is sampled from existing nodes before the source node is sampled. Default value is TRUE.

Value

A list of class rpacontrol with components alpha, beta, gamma, xi, rho, beta.loop and source.first with meanings as explained under 'Arguments'.

Examples

ccontrol <- rpa_control_scenario(alpha = 0.5, beta = 0.5, beta.loop = FALSE)
wdnet_to_igraph  

Converts a `wdnet` object to an `igraph` object

**Description**

Converts a wdnet object to an igraph object

**Usage**

`wdnet_to_igraph(netwk)`

**Arguments**

- `netwk`  
  A `wdnet` object.

**Value**

An `igraph` object.

**Examples**

```r
netwk <- rpanet(nstep = 1e3)
g <- wdnet_to_igraph(netwk)
```
Index

+.rpacontrol, 2
adj_to_wdnet, 3
assortcoef, 4
centrality, 5
clustcoef, 7
cvxr_control, 9
dprewire, 10
dprewire.range, 12
edgelist_to_wdnet, 13
igraph_to_wdnet, 14
is_wdnet, 15
plot.wdnet, 15
print.rpacontrol, 16
print.wdnet, 16
rpa_control_edgeweight, 20
rpa_control_newedge, 20
rpa_control_preference, 21
rpa_control_reciprocal, 24
rpa_control_scenario, 25
rpacontrol, 17
rpanet, 18
summary.rpacontrol(print.rpacontrol), 16
summary.wdnet(print.wdnet), 16
wdnet_to_igraph, 26