Package ‘wdnet’

June 27, 2022

Title Weighted and Directed Networks

Version 0.0.4

Date 2022-06-26

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>,
Zhang, Wang and Yan (2022) <doi:10.1016/j.physa.2021.126438>,
(3) clustering coefficient of weighted and directed networks,
Fagiolo (2007) <doi:10.1103/PhysRevE.76.026107> and
Clemente and Grassi (2018) <doi:10.1016/j.chaos.2017.12.007>,
(4) network rewiring,
(5) preferential attachment network generation.

Depends R (>= 4.2.0)

License GPL (>= 3.0)

Encoding UTF-8

Imports stats, igraph, wdm, Rcpp, rARPACK, Matrix, CVXR

LinkingTo Rcpp, RcppArmadillo

BugReports `https://gitlab.com/wdnetwork/wdnet/-/issues`

URL `https://gitlab.com/wdnetwork/wdnet`

RoxygenNote 7.2.0

NeedsCompilation yes

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

Maintainer Yelie Yuan <yelie.yuan@uconn.edu>

Repository CRAN

Date/Publication 2022-06-27 18:30:02 UTC

R topics documented:

+.rpactl *Add components to the control list*

Description

'+' is used to combine components to control the PA network generation process. Available components are `rpactl.scenario()`, `rpactl.edgeweight()`, `rpactl.newedge()`, `rpactl.preference()` and `rpactl.reciprocal()`.

Usage

```
## S3 method for class 'rpactl'
e1 + e2
```

Arguments

e1	A list of class `rpactl`.
e2	A list of class `rpactl`.

Value

A list of class `rpactl` with components from `e1` and `e2`.

Examples

```
control <- rpactl.scenario(alpha = 0.5, beta = 0.5) +
    rpactl.preference(sparams = c(1, 1, 0, 0, 1),
        tparams = c(0, 0, 1, 1, 1))

control <- rpactl.scenario(alpha = 1) +
    rpactl.edgeweight(distribution = rgamma,
        dparams = list(shape = 5, scale = 0.2),
        shift = 1)
```

adj_to_edge *Convert adjacency matrix to edgelist and edgeweight.*

Description

Convert adjacency matrix to edgelist and edgeweight.

Usage

```
adj_to_edge(adj, directed = TRUE, weighted = TRUE)
```

Arguments

adj	Adjacency matrix of a network.
directed	Logical, whether the network is directed. Passed to `igraph::graph_from_adjacency_matrix`.
weighted	Passed to `igraph::graph_from_adjacency_matrix`. This argument specifies whether to create a weighted graph from an adjacency matrix. If it is NULL then an unweighted graph is created and the elements of the adjacency matrix gives the number of edges between the vertices. If it is TRUE then a weighted graph is created and the name of the edge attribute will be weight.

Value

A list of edgelist and edgeweight.

---

assortcoef *Assortativity coefficient*

Description

Compute the assortativity coefficient of a network.

Usage

```
assortcoef(
  edgelist = NULL,
  edgeweight = NULL,
  adj = NULL,
  directed = TRUE,
  f1 = NULL,
  f2 = NULL
)
```

Arguments

edgelist	A two column matrix represents edges. If `NULL`, `edgelist` and `edgeweight` will be extracted from the adjacency matrix `adj`.
edgeweight	A vector represents the weight of edges. If `edgelist` is provided and `edgeweight` is `NULL`, all the edges will be considered have weight 1.
adj	An adjacency matrix.
directed	Logical. Whether the edges will be considered as directed.
f1	A vector, represents the first feature of existing nodes. Number of nodes = `length(f1)` = `length(f2)`. Defined for directed networks. If `NULL`, out-strength will be used.
f2	A vector, represents the second feature of existing nodes. Defined for directed networks. If `NULL`, in-strength will be used.

Value

Assortativity coefficient for undirected networks, or 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

- Foster, J.G., Foster, D.V., Grassberger, P. and Paczuski, M. (2010). Edge direction and the structure of networks. *Proceedings of the National Academy of Sciences of the United States*, 107(24), 10815–10820.
- Yuan, Y. Zhang, P. and Yan, J. (2021). Assortativity coefficients for weighted and directed networks. *Journal of Complex Networks*, 9(2), cnab017.

Examples

```
set.seed(123)
control <- rpactl.edgeweight(distribution = rgamma,
    dparams = list(shape = 5, scale = 0.2), shift = 0)
netwk <- rpanet(nstep = 10^4, control = control)
result <- assortcoef(netwk$edgelist, edgeweight = netwk$edgeweight, directed = TRUE)
```

`centrality` *Centrality measures*

Description

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

Usage

```
centrality(
  adj = NULL,
  edgelist = NULL,
  edgeweight = NULL,
  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

- `adj` — An adjacency matrix of a weighted and directed network. If NULL, `edgelist` and `edgeweight` will be used to construct the adjacency matrix.
- `edgelist` — A two column matrix, each row represents a directed edge of the network. It will be ignored if `adj` is not NULL.
- `edgeweight` — A vector represents the weight of edges. If `edgelist` is provided and `edgeweight` is `NULL`, all the edges will be considered have weight 1. It will be ignored if `adj` is not NULL.
- `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 distances 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 an 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

- Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271.
- Newman, M.E.J. (2003). The structure and function of complex networks. *SIAM review*, 45(2), 167–256.
- Opsahl, T., Agneessens, F., Skvoretz, J. (2010). Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32, 245–251.
- Zhang, P., Wang, T. and Yan, J. (2022) PageRank centrality and algorithms for weighted, directed networks with applications to World Input-Output Tables. *Physica A: Statistical Mechanics and its Applications*, 586, 126438.
- Zhang, P., Zhao, J. and Yan, J. (2020+) Centrality measures of networks with application to world input-output tables

Examples

```
## Generate a network according to the Erd\"{o}s-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_ER, measure = "degree", degree.control =
list(alpha = 0.8, mode = "in"))
myclose <- centrality(adj_ER, measure = "closeness", closeness.control =
list( alpha = 0.8, mode = "out", method = "harmonic", distance = FALSE))
mywpr <- centrality(adj_ER, measure = "wpr", wpr.control =
list(gamma = 0.85, theta = 0.75))
```

---

closeness_c *Closeness centrality*

Description

Compute the closeness centrality measures of the vertices in a weighted and directed network represented through its adjacency matrix.

Usage

```
closeness_c(
  adj,
  alpha = 1,
  mode = "out",
  method = "harmonic",
  distance = FALSE
)
```

Arguments

`adj`	is an adjacency matrix of a weighted and directed network
`alpha`	is 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 distances or strong connections. The default setting is `FALSE`.

Value

a list of node names and associated closeness centrality measures

Note

Function `closeness_c` 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*.

References

- Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271.
- Newman, M.E.J. (2003). The structure and function of complex networks. *SIAM review*, 45(2), 167–256.
- Opsahl, T., Agneessens, F., Skvoretz, J. (2010). Node centrality in weighted networks: Generalizing 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

---

`clustcoef` *Directed clustering coefficient*

Description

Compute the clustering coefficient of a weighted and directed network.

Usage

```
clustcoef(adj, method = c("Clemente","Fagiolo"), isolates = "zero")
```

Arguments

`adj`	is an adjacency matrix of an weighted and directed network.
`method`	which method used to compute clustering coefficients: Clemente and Grassi (2018) or Fagiolo (2007).
`isolates`	character, defines how to treat vertices with degree zero and one. If "zero", 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 "zero".

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) base 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

- Barrat, A., Barth\'elemy, M., Pastor-Satorras, R. and Vespignani, A. (2004). The architecture of complex weighted networks. *Proceddings of National Academy of Sciences of the United States of America*, 101(11), 3747–3752.
- Clemente, G.P. and Grassi, R. (2018). Directed clustering in weighted networks: A new perspective. *Chaos, Solitons & Fractals*, 107, 26–38.
- Fagiolo, G. (2007). Clustering in complex directed networks. *Physical Review E*, 76, 026107.

Examples

```
## Generate a network according to the Erd\"{o}s-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)
mycc <- clustcoef(adj_ER, method = "Clemente")
system.time(mycc)
```

cvxr.control *Parameters passed to CVXR::solver().*

Description

Defined for the convex optimization problems for solving `eta`. The control list is passed to `dprewire` and `dprewire.range`.

Usage

```
cvxr.control(
  solver = "ECOS",
  ignore_dcp = FALSE,
  warm_start = FALSE,
  verbose = FALSE,
  parallel = FALSE,
  gp = FALSE,
  feastol = NULL,
  reltol = NULL,
  abstol = NULL,
  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.
`reltol`	The relative tolerance on the duality gap.
`abstol`	The absolute tolerance on the duality gap.
`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)
```

---

degree_c — *Degree-based centrality*

Description

Compute the degree centrality measures of the vertices in a weighted and directed network represented through its adjacency matrix.

Usage

```
degree_c(adj, alpha = 1, mode = "out")
```

Arguments

adj	is an adjacency matrix of a weighted and directed network
alpha	is 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.

Value

a list of node names and associated degree centrality measures

Note

Function `degree_c` is an extension of function `strength` in package `igraph` and an alternative of function `degree_w` in package `tnet`. Function `degree_c` uses adjacency matrix as input.

References

- Opsahl, T., Agneessens, F., Skvoretz, J. (2010). Node centrality in weighted networks: Generalizing 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

dprewire *Degree preserving rewiring.*

Description

Rewire a given network to have predetermined assortativity coefficients while preserving node degree.

Usage

```
dprewire(
  edgelist = NULL,
  directed = TRUE,
  adj = NULL,
  target.assortcoef = list(outout = NULL, outin = NULL, inout = NULL, inin = NULL),
  control = list(iteration = 10, nattempts = NULL, history = FALSE, cvxr.control =
    cvxr.control(), eta.obj = function(x) 0),
  eta = NULL
)
```

Arguments

edgelist A two column matrix, each row represents an edge of the network.

directed Logical, whether the network is directed or not.

adj Adjacency matrix of an unweighted network. It will be ignored if edgelist is provided.

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, 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.
- 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 — An 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

There are two steps in this algorithm. It 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. The function only works for unweighted networks.

Each rewiring attempt samples two rows from `edgelist`, for example Edge1:(v_1, v_2) and Edge2:(v_3, v_4). For directed networks, if the rewiring attempt is accepted, the sampled edges are replaced 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 `edgelist`; assortativity coefficient(s) after each iteration; rewiring history (including the index of sampled edges and rewiring result); solved `eta` and its corresponding assortativity coefficient(s), if applicable.

Examples

```
set.seed(123)
edgelist <- rpanet(1e4, control = rpactl.scenario(
   alpha = 0.4, beta = 0.3, gamma = 0.3))$edgelist
## rewire a directed network to have predetermined assortativity coefficients
target.assortcoef <- list("outout" = -0.2, "outin" = 0.2)
ret1 <- dprewire(edgelist, directed = TRUE,
               target.assortcoef = target.assortcoef,
               control = list(iteration = 200))
plot(ret1$assortcoef$Iteration, ret1$assortcoef$"outout")
plot(ret1$assortcoef$Iteration, ret1$assortcoef$"outin")

edgelist <- rpanet(1e4, control = rpactl.scenario(
                  alpha = 0.3, beta = 0.1, gamma = 0.3, xi = 0.3),
                  directed = FALSE)$edgelist
## rewire an undirected network to have predetermined assortativity coefficient
ret2 <- dprewire(edgelist, directed = FALSE, target.assortcoef = 0.3,
               control = list(iteration = 100, eta.obj = CVXR::norm2,
               history = TRUE))
plot(ret2$assortcoef$Iteration, ret2$assortcoef$Value)
```

dprewire.range — *Range of assortativity coefficient.*

Description

The assortativity coefficient of a given network may not achieve all the values within -1 and 1 via degree preserving rewiring. This function computes the range of assortativity coefficients that can be achieved through degree preserving rewiring. The algorithm is designed for unweighted networks.

Usage

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

Arguments

`edgelist`	A two column matrix, each row represents an edge of the network.
`directed`	Logical, whether the network is directed or not.
`adj`	Adjacency matrix of an unweighted network. It will be ignored if `edgelist` 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` with the constraints `target.assortcoef`.
`target.assortcoef`	A list of constraints, it has 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 through convex optimization. The 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

Range of the interested assortativity coefficient; solved `eta` and its corresponding assortativity coefficients.

Examples

```
set.seed(123)
edgelist <- rpanet(5e3, control =
        rpactl.scenario(alpha = 0.5, beta = 0.5))$edgelist
ret1 <- dprewire.range(edgelist, directed = TRUE, which.range = "outin",
        target.assortcoef = list("outout" = c(-0.3, 0.3), "inout" = 0.1))
ret2 <- dprewire(edgelist, eta = ret1$lbound$eta, control = list(iteration = 100))
plot(ret2$assortcoef$Iteration, ret2$assortcoef$"outin")
ret3 <- dprewire(edgelist, eta = ret1$ubound$eta, control = list(iteration = 100))
plot(ret3$assortcoef$Iteration, ret3$assortcoef$"outin")
```

---

dprewire_directed — *Degree preserving rewiring for directed networks*

Description

Degree preserving rewiring towards the target structure `eta`.

Usage

```
dprewire_directed(
  edgelist,
  eta,
  iteration = 1,
  nattempts = NULL,
  rewire.history = FALSE
)
```

Arguments

`edgelist`	A two column matrix, each row represents a directed edge from the first column to the second column.
`eta`	An matrix, target structure eta generated by `wdnet::get_eta_directed()`.
`iteration`	An integer, number of rewiring iterations, each iteration consists of `nattempts` rewiring attempts.
`nattempts`	An integer, number of rewiring attempts for each iteration. Default value equals the number of rows of edgelist.
`rewire.history`	Logical, whether the rewiring history should be returned.

Value

Rewired edgelist, degree based assortativity coefficients after each iteration, rewiring history (including the index of sampled edges and rewiring result). For each rewiring attempt, two rows are sampled form the edgelist, for example Edge1:(v_1, v_2) and Edge2:(v_3, v_4), if the rewiring attempt is accepted, the sampled edges are replaced as (v_1, v_4), (v_3, v_2).

---

dprewire_directed_cpp *Degree preserving rewiring process for directed networks.*

Description

Degree preserving rewiring process for directed networks.

Usage

```
dprewire_directed_cpp(
  iteration,
  nattempts,
  targetNode,
  sourceOut,
  sourceIn,
  targetOut,
  targetIn,
  index_s,
  index_t,
  eta,
  rewire_history
)
```

Arguments

`iteration`	Integer, number of iterations of nattempts rewiring attempts.
`nattempts`	Integer, number of rewiring attempts per iteration.
`targetNode`	Vector, target node sequence - 1.
`sourceOut`	Vector, source nodes' out-degree.
`sourceIn`	Vector, source nodes' in-degree.
`targetOut`	Vector, target nodes' out-degree.
`targetIn`	Vector, target nodes' in-degree.
`index_s`	Index of source nodes' out- and in-degree. index_s/index_t bridges the indices of source/target nodes and the target structure eta.
`index_t`	Index of target nodes' out- and in-degree.
`eta`	Matrix, target structure eta generated by `wdnet::get_eta_directed()`.
`rewire_history`	Logical, whether the rewiring history should be returned.

Value

Target node sequence, four directed assortativity coefficients after each iteration, and rewire history.

`dprewire_undirected` *Degree preserving rewiring for undirected networks*

Description

Degree preserving rewiring towards the target structure `eta`.

Usage

```
dprewire_undirected(
  edgelist,
  eta,
  iteration = 1,
  nattempts = NULL,
  rewire.history = FALSE
)
```

Arguments

`edgelist`	A two column matrix, each row represents an undirected edge.
`eta`	An matrix, target structure `eta` generated by `wdnet::get_eta_undirected()`.
`iteration`	An integer, number of rewiring iterations, each iteration consists of `nattempts` rewiring attempts.
`nattempts`	An integer, number of rewiring attempts for each iteration. Default value equals the number of rows of edgelist.
`rewire.history`	Logical, whether the rewiring history should be returned.

Value

Rewired edgelist, assortativity coefficient after each iteration, and rewiring history (including the index of sampled edges and rewiring result). For each rewiring attempt, two rows are sampled from the edgelist, for example Edge1:{v_1, v_2} and Edge2:{v_3, v_4}, the function try to rewire the sampled edges as {v_1, v_4}, {v_3, v_2} (rewire type 1) or {v_1, v_3}, {v_2, v_4} (rewire type 2) with probability 1/2.

dprewire_undirected_cpp

Degree preserving rewiring process for undirected networks.

Description

Degree preserving rewiring process for undirected networks.

Usage

```
dprewire_undirected_cpp(
  iteration,
  nattempts,
  node1,
  node2,
  degree1,
  degree2,
  index1,
  index2,
  e,
  rewire_history
)
```

Arguments

`iteration`	Integer, number of iterations of nattempts rewiring attempts.
`nattempts`	Integer, number of rewiring attempts per iteration.
`node1`	Vector, first column of edgelist.
`node2`	Vector, second column of edgelist.
`degree1`	Vector, degree of node1 and node2.
`degree2`	Vector, degree of node2 and node1. degree1 and degree2 are used to calculate assortativity coefficient, i.e., degree correlation.
`index1`	Index of the first column of edgelist. index1 and index2 bridge the nodes' degree and the structure e.
`index2`	Index of the second column of edgelist..
`e`	Matrix, target structure e (eta) generated by `wdnet::get_eta_undirected()`.
`rewire_history`	Logical, whether the rewiring history should be returned.

Value

Node sequences, assortativity coefficient after each iteration and rewiring history.

dw_assort *Compute the assortativity coefficient of a weighted and directed network.*

Description

Compute the assortativity coefficient of a weighted and directed network.

Usage

```
dw_assort(adj, type = c("out-in", "in-in", "out-out", "in-out"))
```

Arguments

adj	is an adjacency matrix of a weighted and directed network.
type	which type of assortativity coefficient to compute: "outin" (default), "inin", "outout" or "inout"?

Value

a scalar of assortativity coefficient

Note

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

References

- Foster, J.G., Foster, D.V., Grassberger, P. and Paczuski, M. (2010). Edge direction and the structure of networks. *Proceedings of the National Academy of Sciences of the United States*, 107(24), 10815–10820.
- Yuan, Y. Zhang, P. and Yan, J. (2021). Assortativity coefficients for weighted and directed networks. *Journal of Complex Networks*, 9(2), cnab017.

dw_feature_assort *Feature based assortativity coefficient*

Description

Node feature based assortativity coefficients of a weighted and directed network.

Usage

```
dw_feature_assort(edgelist, edgeweight, f1, f2)
```

Arguments

edgelist	A two column matrix represents edges. If NULL, edgelist and edgeweight will be extracted from the adjacency matrix adj.
edgeweight	A vector represents the weight of edges. If NULL, all the edges are considered have weight 1.
f1	A vector, represents the first feature of existing nodes. Number of nodes = length(f1) = length(f2). Defined for directed networks. If NULL, out-strength will be used.
f2	A vector, represents the second feature of existing nodes. Defined for directed networks. If NULL, in-strength will be used.

Value

Directed weighted assortativity coefficients between source nodes' f1 (or f2) and target nodes' f2(or f1).

Examples

```
set.seed(123)
adj <- matrix(rbinom(400, 1, 0.2) * sample(1:3, 400, replace = TRUE), 20, 20)
f1 <- runif(20)
f2 <- abs(rnorm(20))
ret <- assortcoef(adj = adj, f1 = f1, f2 = f2)
```

---

edge_to_adj *Convert edgelist and edgeweight to adjacency matrix.*

Description

Convert edgelist and edgeweight to adjacency matrix.

Usage

```
edge_to_adj(edgelist, edgeweight = NULL, directed = TRUE)
```

Arguments

edgelist	A two column matrix represents edges.
edgeweight	A vector represents the weight of edges. If NULL, all the edges are considered have weight 1.
directed	Logical, whether the network is directed.

Value

An adjacency matrix.

fill_weight_cpp — *Fill edgeweight into the adjacency matrix. Defined for function* `edge_to_adj`.

Description

Fill edgeweight into the adjacency matrix. Defined for function `edge_to_adj`.

Usage

```
fill_weight_cpp(adj, edgelist, edgeweight)
```

Arguments

`adj`	An adjacency matrix.
`edgelist`	A two column matrix represents the edgelist.
`edgeweight`	A vector represents the weight of edges.

Value

Adjacency matrix with edge weight.

find_node_cpp — *Fill missing nodes in the node sequence. Defined for* `wdnet::rpanet`.

Description

Fill missing nodes in the node sequence. Defined for `wdnet::rpanet`.

Usage

```
find_node_cpp(nodes, edges)
```

Arguments

`nodes`	Source/target nodes, missing nodes are denoted as 0.
`edges`	Sampled edges according to preferential attachment.

Value

Source/target nodes.

`find_node_undirected_cpp` *Fill missing values in node sequence. Defined for* `wdnet::rpanet`.

Description

Fill missing values in node sequence. Defined for `wdnet::rpanet`.

Usage

```
find_node_undirected_cpp(node1, node2, start_edge, end_edge)
```

Arguments

`node1`	Nodes in the first column of edgelist, i.e., `edgelist[, 1]`.
`node2`	Nodes in the second column of edgelist, i.e., `edgelist[, 2]`.
`start_edge`	Index of sampled edges, corresponds to the missing nodes in node1 and node2.
`end_edge`	Index of sampled edges, corresponds to the missing nodes in node1 and node2.

Value

Node sequence.

`get_constr` *Get the constraints for the optimization problem. This function is defined for* `get_eta_directed`.

Description

Get the constraints for the optimization problem. This function is defined for `get_eta_directed`.

Usage

```
get_constr(constrs, target.assortcoef, rho)
```

Arguments

`constrs`	A list of constraints.
`target.assortcoef`	A list of target assortativity levels.
`rho`	A list of variable objects.

Value

A list of constraints.

get_dist — *Get the node-level joint distributions and some empirical distributions with given edgelist.*

Description

Get the node-level joint distributions and some empirical distributions with given edgelist.

Usage

```
get_dist(edgelist = NA, directed = TRUE, joint_dist = FALSE)
```

Arguments

edgelist	A two column matrix represents the directed edges of a network.
directed	Logical, whether the network is directed.
joint_dist	Logical, whether to return edge-level distributions.

Value

A list of distributions and degree vectors.

get_eta_directed — *Compute edge-level distributions for directed networks with respect to desired assortativity level(s).*

Description

Compute edge-level distributions for directed networks with respect to desired assortativity level(s).

Usage

```
get_eta_directed(
  edgelist,
  target.assortcoef = list(outout = NULL, outin = NULL, inout = NULL, inin = NULL),
  eta.obj = function(x) 0,
  which.range = NULL,
  control = cvxr.control()
)
```

Arguments

edgelist — A two column matrix represents the directed edges of a network.

target.assortcoef — List, represents the predetermined value or range of assortativity coefficients.

eta.obj — A convex function of `eta` to be minimized when `which.range` is `NULL`. Defaults to 0.

which.range — Character, "outout", "outin", "inout" or "inin". Represents the interested degree based assortativity coefficient. Default is `NA`.

control — A list of parameters passed to `CVXR::solve()` when solving for `eta` or computing the range of assortativity coefficient.

Value

Assortativity coefficients and joint distributions. If `which.range` is specified, the range of the interested coefficient and the corresponding joint distributions will be returned, provided the predetermined `target.assortcoef` is satisfied.

---

get_eta_undirected — *Compute edge-level distribution for undirected networks with respect to desired assortativity level.*

Description

Compute edge-level distribution for undirected networks with respect to desired assortativity level.

Usage

```
get_eta_undirected(
  edgelist,
  target.assortcoef = NULL,
  eta.obj = function(x) 0,
  control = cvxr.control()
)
```

Arguments

edgelist — A two column matrix represents the undirected edges of a network.

target.assortcoef — Numeric, represents the predetermined assortativity coefficient. If `NA`, the range of assortativity coefficient and corresponding joint distribution are returned.

eta.obj — A convex function of `eta` to be minimized when `target.assortcoef` is not `NA`. Defaults to 0.

control — A list of parameters passed to `CVXR::solve()` when solving for `eta` or computing the range of assortativity coefficient.

Value

Assortativity level and corresponding edge-level distribution.

`get_values` *Get the value of an object from the optimization problem. This function is defined for* `get_eta_directed`*.*

Description

Get the value of an object from the optimization problem. This function is defined for `get_eta_directed`.

Usage

```
get_values(object, result, mydist)
```

Arguments

`object`	An object from the optimization problem.
`result`	A list returned from `CVXR::solve()`.
`mydist`	A list returned from `get_dist()`.

Value

Value of the object.

`node_strength_cpp` *Aggregate edgeweight into nodes' strength.*

Description

Aggregate edgeweight into nodes' strength.

Usage

```
node_strength_cpp(snode, tnode, weight, nnode, weighted = TRUE)
```

Arguments

`snode`	Source nodes.
`tnode`	Target nodes.
`weight`	Edgeweight.
`nnode`	Number of nodes.
`weighted`	Logical, true if the edges are weighted, false if not.

Value

Out-strength and in-strength.

rpactl.edgeweight *Set parameters for controlling weight of new edges*

Description

Set parameters for controlling weight of new edges

Usage

```
rpactl.edgeweight(distribution = NA, dparams = list(), shift = 1)
```

Arguments

distribution	Distribution function for edge weights. Default is NA. If specified, its first argument must be the number of observations.
dparams	Additional parameters passed on to distribution. The name of parameters must be specified.
shift	A constant add to the specified distribution. Default value is 1.

Value

A list of class rpactl with components distribution, dparams, and shift with meanings as explained under 'Arguments'.

Examples

```
# Edge weight follows Gamma(5, 0.2).
control <- rpactl.edgeweight(distribution = rgamma,
    dparams = list(shape = 5, scale = 0.2),
    shift = 0)

# Constant edge weight
control <- rpactl.edgeweight(shift = 2)
```

rpactl.newedge *Set parameters for controlling new edges in each step*

Description

Set parameters for controlling new edges in each step

Usage

```
rpactl.newedge(
  distribution = NA,
  dparams = list(),
  shift = 1,
  snode.replace = TRUE,
  tnode.replace = TRUE,
  node.replace = TRUE
)
```

Arguments

`distribution`	Distribution function for number of new edges. Default is `NA`. If specified, its first argument must be the number of observations.
`dparams`	Additional parameters passed on to `distribution`. The name of parameters must be specified.
`shift`	A constant add to the specified distribution. Default value is 1.
`snode.replace`	Logical, whether the source nodes in the same step should be sampled with replacement. Defined for directed networks.
`tnode.replace`	Logical, whether the target nodes in the same step should be sampled with replacement. Defined for directed networks.
`node.replace`	Logical, 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 `rpactl` with components `distribution`, `dparams`, `shift`, `snode.replace`, `tnode.replace` and `node.replace` with meanings as explained under 'Arguments'.

Examples

```
control <- rpactl.newedge(distribution = rpois,
    dparams = list(lambda = 2),
    shift = 1,
    node.replace = FALSE)
```

`rpactl.preference` *Set parameters for source and target preference function*

Description

Set parameters for source and target preference function

Usage

```
rpactl.preference(
  sparams = c(1, 1, 0, 0, 1),
  tparams = c(0, 0, 1, 1, 1),
  params = c(1, 1)
)
```

Arguments

sparams
: Parameters of the source preference function for directed networks. Probability of choosing an existing node as the source node is proportional to `sparams[1] * out-strength^sparams[2] + sparams[3] * in-strength^sparams[4] + sparams[5]`.

tparams
: Parameters of the target preference function for directed networks. Probability of choosing an existing node as the source node is proportional to `tparams[1] * out-strength^tparams[2] + tparams[3] * in-strength^tparams[4] + tparams[5]`.

params
: Parameters of the preference function for undirected networks. Probability of choosing an existing node is proportional to `strength^param[1] + param[2]`.

Value

A list of class `rpactl` with components `sparams`, `tparams`, and `params` with meanings as explained under 'Arguments'.

Examples

```
control <- rpactl.preference(sparams = c(1, 2, 0, 0, 0.1),
    tparams = c(0, 0, 1, 2, 0.1))
```

---

rpactl.reciprocal *Set parameters for controlling reciprocal edges*

Description

Set parameters for controlling reciprocal edges

Usage

```
rpactl.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`, which belongs to group `i`, to node `B`, which belongs to 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 `rpactl` with components `group.prob`, `recip.prob`, and `selfloop.recip` with meanings as explained under 'Arguments'.

Examples

```
control <- rpactl.reciprocal(group.prob = c(0.4, 0.6),
    recip.prob = matrix(runif(4), ncol = 2))
```

`rpactl.scenario` *Set parameters for controlling the probability of edge scenarios*

Description

Set parameters for controlling the probability of edge scenarios

Usage

```
rpactl.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.loop`	Logical, whether self-loops are allowed under beta scenario. Default value is `TRUE`.

source.first — Logical. 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 rpactl with components alpha, beta, gamma, xi, rho, beta.loop and source.first with meanings as explained under 'Arguments'.

Examples

```
control <- rpactl.scenario(alpha = 0.5, beta = 0.5, beta.loop = FALSE)
```

---

rpanet — *Generate PA networks.*

Description

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

Usage

```
rpanet(
  nstep = 10^3,
  seednetwork = NULL,
  control = NULL,
  directed = TRUE,
  method = c("binary", "naive", "edgesampler", "nodelist")
)
```

Arguments

nstep — Number of steps when generating a network.

seednetwork — A list represents the seed network. If NULL, seednetwork will have one edge from node 1 to node 2 with weight 1. It consists of the following components: a two column matrix edgelist represents the edges; a vector edgeweight represents the weight of edges; an integer vector nodegroup represents the group of nodes. nodegroup is defined for directed networks, if NULL, all nodes from the seed graph are considered from group 1.

control — A list of parameters that controls the PA generation process. The default value is rpactl.scenario() + rpactl.edgeweight() + rpactl.newedge() + rpactl.preference() + rpactl.reciprocal(). Under the default setup, in 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.

directed — Logical, whether to generate directed networks. If FALSE, the edge directions are omitted.

method — Which method to use: binary, naive, edgesampler or nodelist. For nodelist and edgesampler methods, the source preference function must be out-degree (out-strength) plus a nonnegative constant, the target preference function must be in-degree (in-strength) plus a nonnegative constant, beta.loop must be TRUE. Besides, nodelist method only works for unweighted networks, rpactl.edgeweight, rpactl.newedge, rpactl.reciprocal must set as default; node.replace, snode.replace, tnode.replace must be TRUE for edgesampler method.

Value

A list with the following components: edgelist, edgeweight, strength for undirected networks, outstrength and instrength for directed networks, number of new edges in each step newedge (reciprocal edges are not included), control list control, node group nodegroup (if applicable) and edge scenario scenario (1~alpha, 2~beta, 3~gamma, 4~xi, 5~rho, 6~reciprocal). The scenario of edges from seednetwork are denoted as 0.

Note

The nodelist method implements the algorithm from Wan et al. (2017). The edgesampler first samples edges then find the source/target node of the sampled edge. If all the edges are of weight 1, the network can be considered as unweighted, node strength then equals node degree.

References

- Wan P, Wang T, Davis RA, Resnick SI (2017). Fitting the Linear Preferential Attachment Model. Electronic Journal of Statistics, 11(2), 3738–3780.

Examples

```
# Control edge scenario and edge weight through rpactl.scenario()
# and rpactl.edgeweight(), respectively, while keeping rpactl.newedge(),
# rpactl.preference() and rpactl.reciprocal() as default.
set.seed(123)
control <- rpactl.scenario(alpha = 0.5, beta = 0.5) +
    rpactl.edgeweight(distribution = rgamma,
        dparams = list(shape = 5, scale = 0.2), shift = 0)
ret1 <- rpanet(nstep = 1e3, control = control)

# In addition, set node groups and probability of creating reciprocal edges.
control <- control + rpactl.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
# ret2 as a seed network.
control <- control + rpactl.newedge(distribution = rpois,
    dparams = list(lambda = 2), shift = 1)
ret3 <- rpanet(nstep = 1e3, seednetwork = ret2, control = control)
```

rpanet_general *Generate a PA network with non-linear preference functions*

Description

Generate a PA network with non-linear preference functions

Usage

```
rpanet_general(
  nstep,
  seednetwork,
  control,
  directed,
  m,
  sum_m,
  w,
  nnode,
  nedge,
  method,
  sample.recip
)
```

Arguments

nstep	Number of steps when generating a network.
seednetwork	A list represents the seed network. If NULL, seednetwork will have one edge from node 1 to node 2 with weight 1. It consists of the following components: a two column matrix edgelist represents the edges; a vector edgeweight represents the weight of edges; a integer vector nodegroup represents the group of nodes. nodegroup is defined for directed networks, if NULL, all nodes from the seed graph are considered from group 1.
control	A list of parameters that controls the PA generation process. The default value is rpactl.scenario() + rpactl.edgeweight() + rpactl.newedge() + rpactl.preference() + rpactl.reciprocal(). By default, in 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.
directed	Logical, whether to generate directed networks. If FALSE, the edge directions are ignored.
m	Integer vector, number of new edges in each step.
sum_m	Integer, summation of m.
w	Vector, weight of new edges.
nnode	Integer, number of nodes in seednetwork.
nedge	Integer, number of edges in seednetwork.
method	Which method to use when generating PA networks: "binary" or "naive".
sample.recip	Whether reciprocal edges will be added.

Value

A list with the following components: edgelist, edgeweight, strength for undirected networks, outstrength and instrength for directed networks, number of new edges in each step newedge (reciprocal edges are not included), control list control, node group nodegroup (if applicable) and edge scenario scenario (1~alpha, 2~beta, 3~gamma, 4~xi, 5~rho, 6~reciprocal). The scenario of edges from seednetwork are denoted as 0.

---

rpanet_nodelist_cpp   *Preferential attachment algorithm for simple situations, e.g., edge weight equals to 1, number of new edges per step is 1.*

Description

Preferential attachment algorithm for simple situations, e.g., edge weight equals to 1, number of new edges per step is 1.

Usage

```
rpanet_nodelist_cpp(
  snode,
  tnode,
  scenario,
  nnode,
  nedge,
  delta_out,
  delta_in,
  directed
)
```

Arguments

snode	Source nodes.
tnode	Target nodes.
scenario	Sequence of alpha, beta, gamma, xi, rho scenarios.
nnode	Number of nodes in seed network.
nedge	Number of edges in seed network.
delta_out	Tuning parameter.
delta_in	Tuning parameter.
directed	Whether the network is directed.

Value

Number of nodes, sequences of source and target nodes.

rpanet_simple *Generate a PA network with linear preference functions.*

Description

Source preference function must be out-degree (out-strength) plus a nonnegative constant; target preference function must be in-degree (in-strength) plus a nonnegative constant.

Usage

```
rpanet_simple(
  nstep,
  seednetwork,
  control,
  directed,
  m,
  sum_m,
  w,
  ex_node,
  ex_edge,
  method
)
```

Arguments

nstep	Number of steps when generating a network.
seednetwork	A list represents the seed network. If NULL, seednetwork will have one edge from node 1 to node 2 with weight 1. It consists of the following components: a two column matrix edgelist represents the edges; a vector edgeweight represents the weight of edges; a integer vector nodegroup represents the group of each node. nodegroup is defined for directed networks, if NULL, all nodes from the seed graph are considered from group 1.
control	A list of parameters to be used when generate network.
directed	Logical, whether to generate directed networks. When FALSE, the edge directions are omitted.
m	Integer vector, number of new edges in each step.
sum_m	Integer, summation of m.
w	Vector, weight of new edges.
ex_node	Integer, number of nodes in seednetwork.
ex_edge	Integer, number of edges in seednetwork.
method	Which method to use, nodelist or edgesampler.

Value

A list with the following components: edgelist, edgeweight, out- and in-strength, number of edges per step (m), scenario of each new edge (1~alpha, 2~beta, 3~gamma, 4~xi, 5~rho). The edges in the seed graph are denoted as scenario 0.

`rpanet_wan` *Simulating a Preferential Attachment Network*

Description

Simulating a Preferential Attachment Network

Usage

```
rpanet_wan(alpha, beta, gamma, xi, delta_in, delta_out, nedge)
```

Arguments

`alpha`	Scalar probability of adding an edge from the new node to an existing node
`beta`	Scalar probability of adding an edge between two existing nodes.
`gamma`	Scalar probability of adding an edge from an existing node to a new node.
`xi`	Scalar probability of ...
`delta_in`	Growth rate parameter for nodes' instrength
`delta_out`	Growth rate parameter for nodes' outstrength
`nedge`	The number of edges to be generated

Value

A list with the following components: in_degree, out_degree, edge_start, edge_end, evolution

---

`sample_node_cpp` *Uniformly draw a node from existing nodes for each time step. Defined for* `wdnet::rpanet`.

Description

Uniformly draw a node from existing nodes for each time step. Defined for `wdnet::rpanet`.

Usage

```
sample_node_cpp(total_node)
```

Arguments

`total_node`	Number of existing nodes at each time step.

Value

Sampled nodes.

`wdnet` — *wdnet: Weighted and Directed Networks*

Description

This package provides functions to conduct network analysis

- Assortativity, centrality, clustering coefficient for weighted and directed networks
- Rewire an unweighted network with given assortativity coefficient(s)
- Preferential attachment (PA) network generation

Details

The development version of this package is available on Gitlab (https://gitlab.com/wdnetwork/wdnet).

`wpr` — *Weighted PageRank centrality*

Description

Compute the weighted PageRank centrality measures of the vertices in a weighted and directed network represented through its adjacency matrix.

Usage

```
wpr(adj, gamma = 0.85, theta = 1, prior.info)
```

Arguments

`adj`	is an adjacency matrix of a weighted and directed network
`gamma`	is the damping factor; it takes 0.85 (default) if not given.
`theta`	is 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 with corresponding weighted PageRank scores

Note

Function `wpr` is an extension of function `page_rank` in package `igraph`.

References

- Zhang, P., Wang, T. and Yan, J. (2022) PageRank centrality and algorithms for weighted, directed networks with applications to World Input-Output Tables. *Physica A: Statistical Mechanics and its Applications*, 586, 126438.

Index