

Package: netUtils (via r-universe)

October 11, 2026

Title A Collection of Tools for Network Analysis

Version 1.0.0

Description Provides a collection of network analytic (convenience) functions which are missing in other standard packages. This includes triad census with attributes [doi:10.1016/j.socnet.2019.04.003](https://doi.org/10.1016/j.socnet.2019.04.003), core-periphery models [doi:10.1016/S0378-8733\(99\)00019-2](https://doi.org/10.1016/S0378-8733(99)00019-2), and several graph generators. Most functions are build upon 'igraph'.

URL <https://github.com/schochastics/netUtils/>,
<https://schochastics.github.io/netUtils/>

BugReports <https://github.com/schochastics/netUtils/issues>

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

LinkingTo Rcpp

Imports Rcpp, igraph (>= 2.3.0), stats

Suggests covr, GA, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libglpk-dev libxml2-dev

Repository <https://cynkra.r-universe.dev>

Date/Publication 2026-09-30 12:21:08 UTC

RemoteUrl <https://github.com/schochastics/netUtils>

RemoteRef HEAD

RemoteSha d5c07ddc7032d926dbc5f2d81e3b3c6e24d6df4f

Contents

as_adj_list1	2
as_adj_weighted	3
as_multi_adj	4
bipartite_from_data_frame	4
clique_vertex_mat	5
core_periphery	6
dyad_census_attr	7
graph_cartesian	8
graph_cor	9
graph_direct	10
graph_from_multi_edgelist	10
graph_kpartite	11
helpers	12
reciprocity_cor	13
sample_coreseq	14
sample_lfr	15
sample_pa_homophilic	16
split_graph	17
str.igraph	18
structural_equivalence	19
triad_census_attr	19
Index	21

as_adj_list1	<i>Adjacency list</i>
--------------	-----------------------

Description

Create adjacency lists from a graph, either for adjacent edges or for neighboring vertices. This version is faster than the version of igraph but less general.

Usage

```
as_adj_list1(g)
```

Arguments

g An igraph object

Details

The function does not have a mode parameter and returns the same neighbors as as_adj_list(g, mode = "all"), as plain integer vectors. For directed graphs, both in- and out-neighbors are returned.

Value

A list of numeric vectors.

Author(s)

David Schoch

Examples

```
library(igraph)
g <- make_ring(10)
as_adj_list1(g)
```

as_adj_weighted	<i>weighted dense adjacency matrix</i>
-----------------	--

Description

returns the weighted adjacency matrix in dense format

Usage

```
as_adj_weighted(g, attr = NULL)
```

Arguments

g	An igraph object
attr	Either NULL or a character string giving an edge attribute name. If NULL a traditional adjacency matrix is returned. If not NULL then the values of the given edge attribute are included in the adjacency matrix.

Details

This method is faster than as_adj from igraph if you need the weighted adjacency matrix in dense format

Value

Numeric matrix

Author(s)

David Schoch

Examples

```
library(igraph)
g <- sample_gnp(10, 0.2)
E(g)$weight <- runif(ecount(g))
as_adj_weighted(g, attr = "weight")
```

as_multi_adj	<i>Convert a list of graphs to an adjacency matrices</i>
--------------	--

Description

Convenience function that turns a list of igraph objects into adjacency matrices.

Usage

```
as_multi_adj(g_lst, attr = NULL, sparse = FALSE)
```

Arguments

g_lst	A list of igraph object
attr	Either NULL or a character string giving an edge attribute name. If NULL a binary adjacency matrix is returned.
sparse	Logical scalar, whether to create a sparse matrix. The 'Matrix' package must be installed for creating sparse matrices.

Value

List of numeric matrices

Author(s)

David Schoch

bipartite_from_data_frame	<i>two-mode network from a data.frame</i>
---------------------------	---

Description

Create a two-mode network from a data.frame

Usage

```
bipartite_from_data_frame(d, type1, type2, attr = NULL, weighted = TRUE)
```

Arguments

d	data.frame
type1	column name of mode 1
type2	column name of mode 2
attr	named list of edge attributes
weighted	should a weighted graph be created if multiple edges occur

Details

Vertex identifiers are converted to character, so numeric and factor columns are matched by their values. If `weighted = TRUE`, multiple edges are merged, their count is stored in the edge attribute `weight`, numeric edge attributes are summed and other edge attributes keep their first value.

Value

two mode network as igraph object. The vertex attribute `type` is `TRUE` for vertices from `type1` and `FALSE` for vertices from `type2`.

Author(s)

David Schoch

Examples

```
library(igraph)
edges <- data.frame(mode1 = 1:5, mode2 = letters[1:5])
bipartite_from_data_frame(edges, "mode1", "mode2")
```

clique_vertex_mat	<i>Clique Vertex Matrix</i>
-------------------	-----------------------------

Description

Creates the clique vertex matrix with entries (i,j) equal to one if node j is in clique i

Usage

```
clique_vertex_mat(g)
```

Arguments

<code>g</code>	An igraph object
----------------	------------------

Value

Numeric matrix

Author(s)

David Schoch

Examples

```
library(igraph)
g <- sample_gnp(10, 0.2)
clique_vertex_mat(g)
```

core_periphery	<i>Discrete core-periphery model</i>
----------------	--------------------------------------

Description

Fits a discrete core-periphery model to a given network

Usage

```
core_periphery(graph, method = "rk1_dc", iter = 500, ...)
```

Arguments

graph	igraph object
method	algorithm to use (see details)
iter	number of iterations if method=GA
...	other parameters for GA

Details

The function fits the data to an optimal pattern matrix with a genetic algorithm (method="GA") or a rank 1 approximation, either with degree centrality (method="rk1_dc") or eigenvector centrality (method="rk1_ec") . The rank 1 approximation is computationally far cheaper but also more experimental. Best is to compare the results from both models.

Value

list with numeric vector with entries (k1,k2,...ki...) where ki assigns vertex i to either the core (ki=1) or periphery (ki=0), and the maximal correlation with an optimal pattern matrix

Author(s)

David Schoch

References

Borgatti, Stephen P., and Martin G. Everett. "Models of core/periphery structures." *Social networks* 21.4 (2000): 375-395.

Examples

```
set.seed(121)
# split graphs have a perfect core-periphery structure
sg <- split_graph(n = 20, p = 0.3, core = 0.5)
core_periphery(sg)
```

dyad_census_attr	<i>dyad census with node attributes</i>
------------------	---

Description

dyad census with node attributes

Usage

```
dyad_census_attr(g, vattr)
```

Arguments

g	igraph object. should be a directed graph.
vattr	name of vertex attribute to be used.

Details

The node attribute should be integers from 1 to max(attr). Multiple edges and loops are ignored.

Value

dyad census as a data.frame with one row per unordered pair of attribute values from_attr <= to_attr and columns

asym_ab asymmetric dyads with the edge pointing from from_attr to to_attr. For within-group rows (from_attr == to_attr) this is the total number of asymmetric dyads.

asym_ba asymmetric dyads with the edge pointing from to_attr to from_attr. NA for within-group rows, where the direction is not defined.

sym mutual dyads

null empty dyads

Author(s)

David Schoch

Examples

```
library(igraph)
g <- sample_gnp(10, 0.4, directed = TRUE)
V(g)$attr <- c(rep(1, 5), rep(2, 5))
dyad_census_attr(g, "attr")
```

graph_cartesian	<i>Cartesian product of two graphs</i>
-----------------	--

Description

Compute the Cartesian product of two graphs

Usage

```
graph_cartesian(g, h)
```

Arguments

g	An igraph object
h	An igraph object

Details

See https://en.wikipedia.org/wiki/Cartesian_product_of_graphs. The result is undirected and its vertices are named "u-v", where u and v are the names (or ids) of the vertices in g and h.

Value

Cartesian product as igraph object

Author(s)

David Schoch

Examples

```
library(igraph)
g <- make_ring(4)
h <- make_full_graph(2)
graph_cartesian(g, h)
```

graph_cor

*Graph correlation***Description**

This function computes the correlation between networks. Implemented methods expect the graph to be an adjacency matrix or an igraph object.

Usage

```
graph_cor(object1, object2, ...)

## Default S3 method:
graph_cor(object1, object2, ...)

## S3 method for class 'igraph'
graph_cor(object1, object2, diag = FALSE, attr = NULL, ...)

## S3 method for class 'matrix'
graph_cor(object1, object2, diag = FALSE, ...)

## S3 method for class 'array'
graph_cor(object1, object2, diag = FALSE, ...)
```

Arguments

object1	igraph object or adjacency matrix
object2	igraph object or adjacency matrix over the same vertex set as object1
...	additional arguments (currently unused)
diag	logical. Should the diagonal of the adjacency matrices be included? Defaults to FALSE, the standard definition of graph correlation.
attr	Either NULL or a character string giving an edge attribute name used as edge weights for igraph objects. If NULL, the unweighted adjacency matrices are used.

Value

correlation between graphs

graph_direct	<i>Direct product of two graphs</i>
--------------	-------------------------------------

Description

Compute the direct product of two graphs

Usage

```
graph_direct(g, h)
```

Arguments

g	An igraph object
h	An igraph object

Details

See https://en.wikipedia.org/wiki/Tensor_product_of_graphs. The result is undirected and its vertices are named "u-v", where u and v are the names (or ids) of the vertices in g and h.

Value

Direct product as igraph object

Author(s)

David Schoch

Examples

```
library(igraph)
g <- make_ring(4)
h <- make_full_graph(2)
graph_direct(g, h)
```

graph_from_multi_edgelist	<i>Multiple networks from a single edgelist with a typed attribute</i>
---------------------------	--

Description

Create a list of igraph objects from an edgelist according to a type attribute

Usage

```
graph_from_multi_edgelist(
  d,
  from = NULL,
  to = NULL,
  type = NULL,
  weight = NULL,
  directed = FALSE
)
```

Arguments

d	data frame.
from	column name of sender. If NULL, defaults to first column.
to	column of receiver. If NULL, defaults to second column.
type	type attribute to split the edgelist. If NULL, defaults to third column.
weight	optional column name of edge weights. The column is stored as edge attribute weight. Ignored if NULL.
directed	logical scalar, whether or not to create a directed graph.

Value

list of igraph objects.

Author(s)

David Schoch

Examples

```
library(igraph)
d <- data.frame(
  from = rep(c(1, 2, 3), 3), to = rep(c(2, 3, 1), 3),
  type = rep(c("a", "b", "c"), each = 3), weight = 1:9
)
graph_from_multi_edgelist(d, "from", "to", "type", "weight")
```

graph_kpartite	<i>k partite graphs</i>
----------------	-------------------------

Description

Create a complete k-partite graph.

Usage

```
graph_kpartite(n = 10, grp = c(5, 5))
```

Arguments

n	number of nodes
grp	vector of partition sizes. Must sum to n.

Value

igraph object with vertex attribute type giving the partition of each vertex

Author(s)

David Schoch

Examples

```
# 3-partite graph with equal sized groups
graph_kpartite(n = 15, grp = c(5, 5, 5))
```

helpers

helper function

Description

small functions to deal with typical network problems

Usage

```
biggest_component(g)

delete_isolates(g)
```

Arguments

g	igraph object
---	---------------

Value

igraph object

Author(s)

David Schoch

reciprocity_cor	<i>Reciprocity correlation coefficient</i>
-----------------	--

Description

Reciprocity correlation coefficient

Usage

reciprocity_cor(g)

Arguments

g igraph object. should be a directed graph

Details

The usual definition of reciprocity has some defects. It cannot tell the relative difference of reciprocity compared with purely random network with the same number of vertices and edges. The useful information from reciprocity is not the value itself, but whether mutual links occur more or less often than expected by chance.

To overcome this issue, reciprocity can be defined as the correlation coefficient between the entries of the adjacency matrix of a directed graph:

$$\frac{\sum_{i \neq j} (a_{ij} - a')((a_{ji} - a'))}{\sum_{i \neq j} (a_{ij} - a')^2}$$

where a' is the density of g .

This definition gives an absolute quantity which directly allows one to distinguish between reciprocal (>0) and antireciprocal (<0) networks, with mutual links occurring more and less often than random respectively.

Value

Reciprocity as a correlation

Author(s)

David Schoch

References

Diego Garlaschelli; Loffredo, Maria I. (2004). "Patterns of Link Reciprocity in Directed Networks". Physical Review Letters. American Physical Society. 93 (26): 268701

Examples

```
library(igraph)
g <- sample_gnp(20, p = 0.3, directed = TRUE)
reciprocity(g)
reciprocity_cor(g)
```

sample_coreseq	<i>Generate random graphs with a given coreness sequence</i>
----------------	--

Description

Similar to [sample_degseq](#) just with [coreness](#)

Usage

```
sample_coreseq(cores)
```

Arguments

cores	coreness sequence
-------	-------------------

Details

The code is an adaption of the python code from <https://github.com/ktvank/Random-Graphs-with-Prescribed-K-Core-Sequences/>

Value

igraph object of graph with the same coreness sequence as the input

Author(s)

David Schoch

References

Van Koevering, Katherine, Austin R. Benson, and Jon Kleinberg. 2021. 'Random Graphs with Prescribed K-Core Sequences: A New Null Model for Network Analysis'. ArXiv:2102.12604. <https://doi.org/10.1145/3442381.3450001>.

Examples

```
library(igraph)
g1 <- make_graph("Zachary")
kcores1 <- coreness(g1)
g2 <- sample_coreseq(kcores1)
kcores2 <- coreness(g2)

# the sorted arrays are the same
```

```
all(sort(kcores1) == sort(kcores2))
```

sample_lfr

LFR benchmark graphs

Description

Generates benchmark networks for clustering tasks with a priori known communities. The algorithm accounts for the heterogeneity in the distributions of node degrees and of community sizes.

Usage

```
sample_lfr(
  n,
  tau1 = 2,
  tau2 = 1,
  mu = 0.1,
  average_degree,
  max_degree,
  min_community = NULL,
  max_community = NULL,
  on = 0,
  om = 0,
  verbose = FALSE
)
```

Arguments

n	Number of nodes in the created graph.
tau1	Power law exponent for the degree distribution of the created graph. This value must be at least one.
tau2	Power law exponent for the community size distribution in the created graph. This value must be at least one.
mu	Fraction of inter-community edges incident to each node. This value must be in the interval 0 to 1.
average_degree	Desired average degree of nodes in the created graph. This value must be in the interval (0, n] and is required.
max_degree	Maximum degree of nodes in the created graph. This value must be in the interval (0, n] and is required.
min_community	Minimum size of communities in the graph. Either both or none of min_community and max_community must be specified. If none are specified, the community size range is set automatically to $[\max(k_{\min}, 3), \max_degree]$, where $k_{\min}$ is the minimum degree implied by average_degree, max_degree and tau1.
max_community	Maximum size of communities in the graph. Must be at least min_community.

on number of overlapping nodes (a non-negative integer not larger than n).
om number of memberships of the overlapping nodes. Must be at least 2 if on > 0.
verbose logical. Should progress messages of the generator be printed?

Details

code adapted from https://github.com/synwalk/synwalk-analysis/tree/master/lfr_generator.
Random numbers are drawn from R's random number generator, so results can be reproduced with `set.seed()`.

Value

an igraph object with two vertex attributes: membership, an integer vector holding the (first) community of each vertex, and memberships, a list holding all communities of each vertex (only overlapping vertices have more than one).

References

A. Lancichinetti, S. Fortunato, and F. Radicchi.(2008) Benchmark graphs for testing community detection algorithms. Physical Review E, 78. arXiv:0805.4770

Examples

```
# Simple Girven-Newman benchmark graphs
g <- sample_lfr(
  n = 128, average_degree = 16,
  max_degree = 16, mu = 0.1,
  min_community = 32, max_community = 32
)
```

sample_pa_homophilic *Homophilic random graph using BA preferential attachment model*

Description

A graph of n nodes is grown by attaching new nodes each with m edges that are preferentially attached to existing nodes with high degree, depending on the homophily parameters.

Usage

```
sample_pa_homophilic(
  n,
  m,
  minority_fraction,
  h_ab,
  h_ba = NULL,
  directed = FALSE
)
```

Arguments

n	number of nodes
m	number of edges a new node is connected to
minority_fraction	fraction of nodes that belong to the minority group
h_ab	probability to connect a node from group a with group b
h_ba	probability to connect a node from group b with group a. If NULL, h_ab is used.
directed	should a directed network be created

Details

The code is an adaption of the python code from <https://github.com/gesiscss/HomophilicNtwMinorities/>

Value

igraph object

Author(s)

David Schoch

References

Karimi, F., Génois, M., Wagner, C., Singer, P., & Strohmaier, M. (2018). Homophily influences ranking of minorities in social networks. *Scientific reports*, 8(1), 1-12. (<https://www.nature.com/articles/s41598-018-29405-7>)

Espín-Noboa, L., Wagner, C., Strohmaier, M., & Karimi, F. (2022). Inequality and inequity in network-based ranking and recommendation algorithms. *Scientific reports*, 12(1), 1-14. (<https://www.nature.com/articles/s41598-022-05434-1>)

Examples

```
# maximally heterophilic network
sample_pa_homophilic(n = 50, m = 2, minority_fraction = 0.2, h_ab = 1)
# maximally homophilic network
sample_pa_homophilic(n = 50, m = 2, minority_fraction = 0.2, h_ab = 0)
```

split_graph

split graph

Description

Create a random split graph with a perfect core-periphery structure.

Usage

```
split_graph(n, p, core)
```

Arguments

n	number of nodes
p	probability of peripheral nodes to connect to the core nodes
core	fraction of nodes in the core

Value

igraph object

Author(s)

David Schoch

Examples

```
# split graph with 20 nodes and a core size of 10
split_graph(n = 20, p = 0.4, 0.5)
```

str.igraph

Print graphs to terminal

Description

Prints an igraph object to terminal (different than the standard igraph method)

Usage

```
## S3 method for class 'igraph'
str(object, ...)
```

Arguments

object	An igraph object
...	additional arguments to print (ignored)

Value

str does not return anything. The obvious side effect is output to the terminal.

Author(s)

David Schoch

`structural_equivalence`*Maximal Structural Equivalence*

Description

Calculates structural equivalence for an undirected graph

Usage

```
structural_equivalence(g)
```

Arguments

`g` An igraph object

Details

Two nodes `u` and `v` are structurally equivalent if they have exactly the same neighbors. The equivalence classes produced with this function are either cliques or empty graphs.

Value

vector of equivalence classes

Author(s)

David Schoch

`triad_census_attr`*triad census with node attributes*

Description

triad census with node attributes

Usage

```
triad_census_attr(g, vattr)
```

Arguments

`g` igraph object. should be a directed graph
`vattr` name of vertex attribute to be used

Details

The node attribute should be integers from 1 to $\max(\text{attr})$. The output is a named vector where the names are of the form Txxx-abc, where xxx corresponds to the standard triad census notation and "abc" are the attributes of the involved nodes. If there are more than nine attribute values, the attributes are separated by dots (Txxx-a.b.c). For cyclic triads (030C) with three distinct attributes, the two orientations are counted separately: "abc" if the edges point from the smallest to the middle attribute value, "cba" otherwise. Multiple edges and loops are ignored.

The implemented algorithm is comparable to the algorithm in Lienert et al.

Value

triad census with node attributes

Author(s)

David Schoch

References

Lienert, J., Koehly, L., Reed-Tsochas, F., & Marcum, C. S. (2019). An efficient counting method for the colored triad census. *Social Networks*, 58, 136-142.

Examples

```
library(igraph)
set.seed(112)
g <- sample_gnp(20, p = 0.3, directed = TRUE)
# add a vertex attribute
V(g)$type <- rep(1:2, each = 10)
triad_census_attr(g, "type")
```

Index

`as_adj_list1`, [2](#)
`as_adj_weighted`, [3](#)
`as_multi_adj`, [4](#)

`biggest_component` (helpers), [12](#)
`bipartite_from_data_frame`, [4](#)

`clique_vertex_mat`, [5](#)
`core_periphery`, [6](#)
`coreness`, [14](#)

`delete_isolates` (helpers), [12](#)
`dyad_census_attr`, [7](#)

`graph_cartesian`, [8](#)
`graph_cor`, [9](#)
`graph_direct`, [10](#)
`graph_from_multi_edgelist`, [10](#)
`graph_kpartite`, [11](#)

`helpers`, [12](#)

`reciprocity_cor`, [13](#)

`sample_coreseq`, [14](#)
`sample_degseq`, [14](#)
`sample_lfr`, [15](#)
`sample_pa_homophilic`, [16](#)
`set.seed()`, [16](#)
`split_graph`, [17](#)
`str.igraph`, [18](#)
`structural_equivalence`, [19](#)

`triad_census_attr`, [19](#)