Package ‘netdiffuseR’

March 26, 2019

Title Analysis of Diffusion and Contagion Processes on Networks

Version 1.20.2

Date 2019-03-25

Description Empirical statistical analysis, visualization and simulation of
diffusion and contagion processes on networks. The package implements algorithms
for calculating network diffusion statistics such as transmission rate, hazard
rates, exposure models, network threshold levels, infectiousness (contagion),
and susceptibility. The package is inspired by work published in Valente,

Depends R (>= 3.1.1)

License MIT + file LICENSE

LazyData true

Imports Rcpp (>= 0.12.1), sna, network, networkDynamic, Matrix, MASS,
MatchIt, SparseM, methods, grDevices, graphics, stats, utils,
boot, igraph, viridisLite

Suggests covr, testthat, knitr, ape, RSiena, survival

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 6.1.1

Encoding UTF-8

URL https://github.com/USCCANA/netdiffuseR,
https://USCCANA.github.io/netdiffuseR

BugReports https://github.com/USCCANA/netdiffuseR/issues

Classification/MSC 90C35, 90B18, 91D30

Collate 'RcppExports.R' 'imports.r' 'graph_data.r' 'adjmat.r' 'bass.r'
'bootnet.r' 'data.r' 'diffnet-c.R' 'diffnet-class.r'
'diffnet-indexing.r' 'diffnet-methods.r' 'egonets.R'
'formula.r' 'igraph.r' 'infect_suscept.r' 'mentor.r' 'misc.r'
R topics documented:

'moran.r'  'netmatch.r'  'network.r'  'options.R'  'package-doc.r'
'plot_diffnet2.r'  'rewire.r'  'random_graph.R'  'rdiffnet.r'
'read_write_foreign.r'  'select_egoalter.R'  'spatial.R'
'spatial_graph.R'  'struct_equiv.R'  'struct_test.R'
'survey_to_diffnet.R'

NeedsCompilation yes

Author George Vega Yon [aut, cre] (<https://orcid.org/0000-0002-3171-0844>),
  Rewrite functions with Rcpp, plus new features,
Thomas Valente [aut, cph] (<https://orcid.org/0000-0002-8824-5816>), R
  original code),
Stephanie Dyal [ctb] (Package's first version),
Timothy Hayes [ctb] (Package's first version)

Maintainer George Vega Yon <g.vegayon@gmail.com>

Repository CRAN

Date/Publication 2019-03-26 06:00:03 UTC

R topics documented:

approx_geodesic ........................................... 4
as.array.diffnet ........................................... 5
as_dgCMatrix .............................................. 6
bass ......................................................... 8
bootnet ...................................................... 10
brfarmers .................................................... 12
brfarmersDiffNet ........................................... 17
c.diffnet .................................................... 18
classify_adopters ......................................... 19
classify_graph ............................................. 22
cumulative_adopt_count .................................. 23
dgr .......................................................... 24
diag_expand ................................................. 25
diffnet-arithmetic ....................................... 27
diffnet-class .............................................. 28
diffnet_check_attr_class ................................ 34
diffnet_index .............................................. 34
diffreg ...................................................... 37
diffusion-data ............................................. 38
diffusionMap ............................................... 40
drawColorKey .............................................. 43
d扼gelist_to_adjmat ...................................... 44
d扼gess_coords ............................................ 47
d扼gnetAttrs ............................................... 49
d扼g_variance .............................................. 52
dexposure ................................................... 53
fakeDynEdgelist .......................................... 57
fakeEdgelist .............................................. 58
R topics documented:

- fakesurvey ........................................... 58
- fakesurveyDyn ....................................... 59
- grid_distribution ..................................... 60
- hazard_rate .......................................... 61
- igraph .................................................. 63
- infection ............................................. 64
- isolated ............................................... 66
- kfamily ............................................... 68
- kfamilyDiffNet ....................................... 80
- matrix_compare ....................................... 81
- medInnovations ...................................... 83
- medInnovationsDiffNet ................................ 85
- mentor_matching ..................................... 86
- moran .................................................. 88
- netdiffuseR ........................................... 90
- netdiffuseR-graphs ................................... 90
- netdiffuseR-options .................................. 91
- netmatch .............................................. 92
- network ............................................... 94
- nvertices ............................................. 96
- permute_graph ....................................... 97
- plot.diffnet .......................................... 98
- plot_adaptors ........................................ 100
- plot_diffnet ......................................... 101
- plot_diffnet2 ........................................ 103
- plot_infectsuscep .................................... 105
- plot_threshold ....................................... 107
- pretty_within ........................................ 110
- rdiffnet .............................................. 111
- read_pajek ........................................... 114
- read_ucinet_head ..................................... 115
- recode ................................................ 116
- rescale_vertex_igraph ................................. 117
- rewire_graph ........................................ 119
- rgraph_ba ........................................... 123
- rgraph_er ............................................ 125
- rgraph_ws ............................................ 127
- ring_lattice ......................................... 128
- round_to_seq ........................................ 129
- select_egoalter ..................................... 130
- struct_equiv ........................................ 131
- struct_test .......................................... 133
- summary.diffnet ..................................... 137
- survey_to_diffnet ................................... 138
- threshold ............................................ 141
- toa_diff ............................................. 143
- toa_mat .............................................. 144
- transformGraphBy .................................... 145
approx_geodesic

Description
Computes approximate geodesic distance matrix using graph powers and keeping the amount of memory used low.

Usage
approx_geodesic(graph, n = 6L, warn = FALSE)
approx_geodist(graph, n = 6L, warn = FALSE)

Arguments
- **graph**: Any class of accepted graph format (see netdiffuserR-graphs).
- **n**: Integer scalar. Degree of approximation. Bigger values increase precision (see details).
- **warn**: Logical scalar. When TRUE, it warns if the algorithm performs less steps than required.

Details
While both igraph and sna offer very good and computationally efficient routines for computing geodesic distances, both functions return dense matrices, i.e. not sparse, which can be troublesome. Furthermore, from the perspective of social network analysis, path lengths of more than 6 steps, for example, may not be meaningful, or at least, relevant for the researcher. In such cases, approx_geodesic serves as a solution to this problem, computing geodesics up to the number of steps, n, desired, hence, if n = 6, once the algorithm finds all paths of 6 or less steps it will stop, returning a sparse matrix with zeros for those pairs of vertices for which it was not able to find a path with less than n steps.

Depending on the graph size and density, approx_geodesic’s performance can be compared to that of sna::geodist. Although, as n increases, geodist becomes a better alternative.

The algorithm was implemented using power graphs. At each iteration i the power graph of order i is computed, and its values are compared to the current values of the geodesic matrix (which is initialized in zero).

1. Initialize the output ans(n, n)
2. For i=1 to i < n do
(a) Iterate through the edges of $G^i$, if ans has a zero value in the corresponding row+column, replace it with $i$
(b) next
3. Replace all diagonal elements with a zero and return.

This implementation can be more memory efficient than the aforementioned ones, but at the same time it can be significant slower.
approx_geodist is just an alias for approx_geodesic.

Value
A sparse matrix of class \texttt{dgCMatrix} of size $\text{n.nodes(graph)} \times 2$ with geodesic distances up to $n$.

Examples

```r
# A very simple example -------------------------------
g <- ring_lattice(10, 3)
approx_geodesic(g, 6)
sna::geodist(as.matrix(g))[[2]]
igraph::distances(
  igraph::graph_from_adjacency_matrix(g, mode = "directed"),
  mode = "out"
)
```

---

### as.array.diffnet

Coerce a diffnet graph into an array

**Description**

Coerce a diffnet graph into an array

**Usage**

```r
## S3 method for class 'diffnet'
as.array(x, ...)
```

**Arguments**

- `x` A diffnet object.
- `...` Ignored.

**Details**

The function takes the list of sparse matrices stored in `x` and creates an array with them. Attributes and other elements from the diffnet object are dropped. `dimnames` are obtained from the metadata of the diffnet object.
Value

A three-dimensional array of \( T \) matrices of size \( n \times n \).

See Also

diffnet.

Other diffnet methods: %*%, c.diffnet, diffnet-arithmetic, diffnet-class, diffnet_index, plot.diffnet, summary.diffnet

Examples

# Creating a random diffnet object
set.seed(84117)
mydiffnet <- rdiffnet(30, 5)

# Coercing it into an array
as.array(mydiffnet)

---

**as_dgCMatrix**

Coerce a matrix-like objects to dgCMatrix (sparse matrix)

Description

This helper function allows easy coercion to sparse matrix objects from the *Matrix* package, *dgCMatrix*.

Usage

```r
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```

```r
as.dgCMatrix(x, make.dimnames = TRUE, ...)
```

```r
as_spmat(x, make.dimnames = TRUE, ...)
```

## Default S3 method:
```
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```

## S3 method for class 'diffnet'
```
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```

## S3 method for class 'array'
```
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```

## S3 method for class 'igraph'
```
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```

## S3 method for class 'network'
```
as_dgCMatrix(x, make.dimnames = TRUE, ...)
```
as_dgCMatrix

as_dgCMatrix(x, make.dimnames = TRUE, ...)

## S3 method for class 'list'
as_dgCMatrix(x, make.dimnames = TRUE, ...)

Arguments

x An object to be coerced into a sparse matrix.
make.dimnames Logical scalar. When TRUE, it makes sure that the returned object has dimnames.
... Further arguments passed to the method.

Details

In the case of the igraph and network methods, ... is passed to as_adj and as.matrix.network respectively.

Value

Either a list with dgCMatrix objects or a dgCMatrix object.

Examples

set.seed(1231)
x <- rgraph_er(10)

# From matrix object
as_dgCMatrix(as.matrix(x))

# From a network object
as_dgCMatrix(network::as.network(as.matrix(x)))

# From igraph object
as_dgCMatrix(igraph::graph_from_adjacency_matrix(x))

# From array
myarray <- array(dim=c(10,10,2))
myarray[,1] <- as.matrix(x)
myarray[,2] <- as.matrix(x)

myarray
as_dgCMatrix(myarray)

# From a diffnet object
ans <- as_dgCMatrix(medInnovationsDiffNet)
str(ans)
Bass Model

Description

Fits the Bass Diffusion model. In particular, fits an observed curve of proportions of adopters to $F(t)$, the proportion of adopters at time $t$, finding the corresponding coefficients $p$, innovation rate, and $q$, imitation rate.

Usage

```r
fitbass(dat, ...)
```

```r
## S3 method for class 'diffnet'
fitbass(dat, ...)
```

```r
## Default S3 method:
fitbass(dat, ...)
```

```r
## S3 method for class 'diffnet_bass'
plot(x, y = 1:length(x$m$1hs()), add = FALSE,
     pch = c(21, 24), main = "Bass Diffusion Model",
     ylab = "Proportion of adopters", xlab = "Time", type = c("b", "b"),
     lty = c(2, 1), col = c("black", "black"), bg = c("lightblue",
     "gray"), include.legend = TRUE, ...)
```

```r
bass_F(Time, p, q)
```

```r
bass_dF(p, q, Time)
```

```r
bass_f(Time, p, q)
```

Arguments

- `dat` Either a `diffnet` object, or a numeric vector. Observed cumulative proportion of adopters.
- `...` Further arguments passed to the method.
- `x` An object of class `diffnet_bass`.
- `y` Integer vector. Time (label).
- `add` Passed to `matplot`.
- `pch` Passed to `matplot`.
- `main` Passed to `matplot`.
- `ylab` Character scalar. Label of the y axis.
- `xlab` Character scalar. Label of the x axis.
bass

Among the arguments passed to the function:

- **type**
  Passed to `matplot`. 
- **lty**
  Passed to `matplot`. 
- **col**
  Passed to `matplot`. 
- **bg**
  Passed to `matplot`. 
- **include.legend**
  Logical scalar. When `TRUE`, draws a legend. 
- **time**
  Integer vector with values greater than 0. The `t` parameter. 
- **p**
  Numeric scalar. Coefficient of innovation. 
- **q**
  Numeric scalar. Coefficient of imitation.

### Details

The function fits the bass model with parameters \([p, q]\) for values \(t = 1, 2, \ldots, T\), in particular, it fits the following function:

\[
F(t) = \frac{1 - \exp(-(p + q)t)}{1 + \frac{q}{p} \exp(-(p + q)t)}
\]

Which is implemented in the `bass_F` function. The proportion of adopters at time \(t\), \(f(t)\) is:

\[
f(t) = \begin{cases} 
F(t), & t = 1 \\
F(t) - F(t - 1), & t > 1 
\end{cases}
\]

and it’s implemented in the `bass_f` function.

For testing purposes only, the gradient of \(F\) with respect to \(p\) and \(q\) is implemented in `bass_dF`.

The estimation is done using `nls`.

### Value

An object of class `nls` and `diffnet_bass`. For more details, see `nls` in the `stats` package.

### Author(s)

George G. Vega Yon

### References


### See Also

Other statistics: `classify_adopters`, `cumulative_adopt_count`, `dgr`, `ego_variance`, `exposure`, `hazard_rate`, `infection`, `moran`, `struct_equiv`, `threshold`, `vertex_covariate_dist`
Examples

# Fitting the model for the Brazilian Farmers Data ------------------------
data(brfarmersDiffNet)
ans <- fitbass(brfarmersDiffNet)

# All the methods that work for the -nls- object work here
ans
summary(ans)
coef(ans)
vcov(ans)

# And the plot method returns both, fitted and observed curve
plot(ans)

---

bootnet  Network Bootstrapping

Description

Implements the bootstrapping method described in Snijders and Borgatti (1999). This function is essentially a wrapper of boot.

Usage

resample_graph(graph, self = NULL, useR = FALSE, ...)

bootnet(graph, statistic, R, resample.args = list(self = FALSE), ...)

## S3 method for class 'diffnet_bootnet'
c(..., recursive = FALSE)

## S3 method for class 'diffnet_bootnet'
print(x, ...)

## S3 method for class 'diffnet_bootnet'
hist(x,
   main = "Empirical Distribution of Statistic",
   xlab = expression(Values ~ of ~ t), breaks = 20, annotated = TRUE,
   b0 = expression(atop(plain("" %up% plain("")), t[0]),
   b = expression(atop(plain("" %up% plain("")), t[]), ask = TRUE,
   ...)

Arguments

graph Any class of accepted graph format (see netdiffuseR-graphs).
self Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
user

Logical scalar. When TRUE, autolinks are filled using an R based routine. Otherwise it uses the Rcpp implementation (default). This is intended for testing only.

... Further arguments passed to the method (see details).

statistic A function that returns a vector with the statistic(s) of interest. The first argument must be the graph, and the second argument a vector of indices (see details)

R Number of reps

resample.args List. Arguments to be passed to resample_graph

recursive Ignored

x A diffnet_bootnet class object.

main Character scalar. Title of the histogram.

xlab Character scalar. x-axis label.

breaks Passed to hist.

annotated Logical scalar. When TRUE marks the observed data average and the simulated data average.

b0 Character scalar. When annotated=TRUE, label for the value of b0.

b Character scalar. When annotated=TRUE, label for the value of b.

ask Logical scalar. When TRUE, asks the user to type <Enter> to see each plot (as many as statistics where computed).

Details

Just like the boot function of the boot package, the statistic that is passed must have as arguments the original data (the graph in this case), and a vector of indices. In each repetition, the graph that is passed is a resampled version generated as described in Snijders and Borgatti (1999).

When self = FALSE, for pairs of individuals that haven been drawn more than once the algorithm, in particular, resample_graph, takes care of filling these pseudo autolinks that are not in the diagonal of the network. By default it is assumed that these pseudo-autolinks depend on whether the original graph had any, hence, if the diagonal has any non-zero value the algorithm assumes that self = TRUE, skipping the ‘filling algorithm’. It is important to notice that, in order to preserve the density of the original network, when assigning an edge value to a pair of the form (i, i) (pseudo-autolinks), such is done with probability proportional to the density of the network, in other words, before choosing from the existing list of edge values, the algorithm decides whether to set a zero value first.

The vector of indices that is passed to statistic, an integer vector with range 1 to n, corresponds to the drawn sample of nodes, so the user can, for example, use it to get a subset of a data.frame that will be used with the graph.

Value

A list of class diffnet_bootnet containing the following:

graph The graph passed to bootnet.
p.value  The resulting p-value of the test (see details).
t0      The observed value of the statistic.
mean_t  The average value of the statistic applied to the simulated networks.
var_t   A vector of length length(t0). Bootstrap variances.
R       Number of simulations.
statistic The function statistic passed to bootnet.
boot    A boot class object as return from the call to boot.
resample.args The list resample.args passed to bootnet.

References

See Also
Other Functions for inference: moran, struct_test

Examples
# set.seed(13)
g <- rgraph_ba(t=99)
ans <- bootnet(g, function(w, ...) length(w@x), R=100)
ans

Brazilian Farmers

Description
From Valente (1995) “In the mid-1960s, Rogers and others conducted an ambitious ‘three country study’ to determine influences on adoption of farm practices in Nigeria, India and Brazil. [...] Only in Brazil, and only for hybrid corn, did adoption of the innovation reach more than a small proportion of the farmers.”

Usage
brfarmers
Format

A data frame with 692 rows and 148 columns:

- **village**: village number
- **idold**: respondent id
- **age**: respondent’s age
- **liveout**: Lived outside of community
- **visits**: # of visits to large city
- **contact**: # of contacts with relatives
- **coop**: membership in coop
- **orgs**: membership in organizations
- **patry**: Patriarchalism score
- **liter**: Literate
- **news1**: # of newspapers or mags pr mon
- **subs**: subscribe to news
- **radio1**: Own radio
- **radio2**: Frequency radio listening
- **radio3**: program preference
- **tv**: frequency Tv viewing
- **movie**: freq movie attendance
- **letter**: freq letter writing
- **source**: total # of sources used for ag
- **practA**: Ever used practice A
- **practB**: Ever used practice B
- **practC**: Ever used practice C
- **practD**: Ever used practice D
- **practE**: Ever used practice E
- **practF**: Ever used practice F
- **practG**: Ever used practice G
- **practH**: Ever used practice H
- **practI**: Ever used practice I
- **practJ**: Ever used practice J
- **practK**: Ever used practice K
- **practL**: Ever used practice L
- **yrA**: A year of adoption
- **yrB**: B year of adoption
- **yrC**: C year of adoption
- **yrD**: D year of adoption
<table>
<thead>
<tr>
<th>yrE</th>
<th>E year of adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>yrF</td>
<td>F year of adoption</td>
</tr>
<tr>
<td>yrG</td>
<td>G year of adoption</td>
</tr>
<tr>
<td>yrH</td>
<td>H year of adoption</td>
</tr>
<tr>
<td>yrI</td>
<td>I year of adoption</td>
</tr>
<tr>
<td>yrJ</td>
<td>J year of adoption</td>
</tr>
<tr>
<td>yrK</td>
<td>K year of adoption</td>
</tr>
<tr>
<td>yrL</td>
<td>L year of adoption</td>
</tr>
<tr>
<td>curA</td>
<td>A Current use</td>
</tr>
<tr>
<td>curB</td>
<td>B Current use</td>
</tr>
<tr>
<td>curC</td>
<td>C Current use</td>
</tr>
<tr>
<td>curD</td>
<td>D Current use</td>
</tr>
<tr>
<td>curE</td>
<td>E Current use</td>
</tr>
<tr>
<td>curF</td>
<td>F Current use</td>
</tr>
<tr>
<td>curG</td>
<td>G Current use</td>
</tr>
<tr>
<td>curH</td>
<td>H Current use</td>
</tr>
<tr>
<td>curI</td>
<td>I Current use</td>
</tr>
<tr>
<td>curJ</td>
<td>J Current use</td>
</tr>
<tr>
<td>curK</td>
<td>K Current use</td>
</tr>
<tr>
<td>curL</td>
<td>L Current use</td>
</tr>
<tr>
<td>src1</td>
<td>Source of aware in A</td>
</tr>
<tr>
<td>timeA</td>
<td>Years ago 1st aware</td>
</tr>
<tr>
<td>src2</td>
<td>Source of more info on A</td>
</tr>
<tr>
<td>src3</td>
<td>Most influential source</td>
</tr>
<tr>
<td>use</td>
<td>use during trial stage</td>
</tr>
<tr>
<td>total</td>
<td>total # of practices adopted</td>
</tr>
<tr>
<td>futatt</td>
<td>Future attitude</td>
</tr>
<tr>
<td>achiev</td>
<td>Achievement Score</td>
</tr>
<tr>
<td>atcred</td>
<td>Attitude toward credit</td>
</tr>
<tr>
<td>littest</td>
<td>Score on functional literacy</td>
</tr>
<tr>
<td>acarcomm</td>
<td>Communication with ACAR repres</td>
</tr>
<tr>
<td>econk</td>
<td>Economic knowledge</td>
</tr>
<tr>
<td>caact</td>
<td>recognize any change agent act</td>
</tr>
<tr>
<td>hfequip</td>
<td># of home &amp; farm equips owned</td>
</tr>
<tr>
<td>politk</td>
<td>political knowledge score</td>
</tr>
<tr>
<td>income</td>
<td>income</td>
</tr>
<tr>
<td>land1</td>
<td>total land area in pasture</td>
</tr>
</tbody>
</table>
land2  total land area planted
cows  # of cows giving milk
land3  total land owned
respf  respondent named as friend
respa  respondent named as ag adv
resppa  respondent named for practic A
resppb  respondent named for practic B
resppe  respondent named for practic C
poly  polymorphic OL for 3 practices
respl  respondent named for loan
resppi  resp named for price info
repscep  resp named for coop comm proj
counter  counterfactuality score
opinion  opinionness score
school  years of schooling by resp
pk1  political know 1
pk2  political know 2
pk3  political know 3
pk4  political know 4
pk5  political know 5
innovtim  innovativeness time
adoptpct  adoption percent
discon  # of practices discontinued
mmcred  Mass media credibility
trust  Trust
stusinc  Status inconsistency
nach  N achievement motivation
attcred2  Attitude toward credit
risk  Risk taking
socpart  Social participate
patriarc  patriarchy
credit2  attit to credit for product
visicit  visitin cities
nondep  non-dependence on farming
oltotal  OL total 7 items t-score
innov  overall innovativeness score
icosmo  cosmo index
immexp  mass media exposure index
iempath  empathy index
iach5    achievement motivation index 5
iach7    achievement motivation index 7
ipk      political knowledge index
immc     mass media credibility index
iol      OL index
yr       Actual Year of Adoption
fs       — MISSING INFO —
ado      Time of Adoption
tri      Triangular values used as appro
hlperc   high low percent of diffusion
hlperc1  — MISSING INFO —
new      new or old villages
card1    card number
sour1    Source: radio
sour2    Source: TV
sour3    Source: Newspaper
sour4    Source: Magazine
sour5    Source: ACAR Bulletin
sour6    Source: Agronomist
sour7    Source: Neighbor
sourc6   — MISSING INFO —
adopt    — MISSING INFO —
net31    nomination friend 1
net32    nomination friend 2
net33    nomination friend 3
net21    nomination influential 1
net22    nomination influential 2
net23    nomination influential 3
net11    nomination practice A
net12    nomination practice B
net13    nomination practice C
net41    nomination coop comm proj
id       — MISSING INFO —
commun   Number of community
toa      Time of Adoption
test     — MISSING INFO —
study    Number of study in Valente (1995)
**Details**

The dataset has 692 respondents (farmers) from 11 communities. Collected during 1966, it spans 20 years of farming practices.

**Source**

The Brazilian Farmers data were collected as part of a USAID-funded study of farming practicing in the three countries, India, Nigeria, and Brazil. There was only one wave of data that contained survey questions regarding social networks, and only in Brazil did diffusion of the studied farming innovations reach an appreciable saturation level- that was for hybrid seed corn. The data were stored along with hundreds of other datasets by the University of Wisconsin library and I, Tom Valente, paid a fee to have the disks mailed to me in the early 1990s.

**References**


**See Also**

Other diffusion datasets: brfarmersDiffNet, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, kfamily, medInnovationsDiffNet, medInnovations

---

**Description**

A directed dynamic graph with 692 vertices and 21 time periods. The attributes in the graph are static and described in brfarmers.

**Format**

A `diffnet` class object.

**See Also**

Other diffusion datasets: brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, kfamily, medInnovationsDiffNet, medInnovations
c.diffnet  

---

**c.diffnet**  

*Combine diffnet objects*

---

**Description**

Combining `diffnet` objects that share time periods and attributes names, but vertices ids (only valid for `diffnet` objects that have an empty intersection between vertices ids).

**Usage**

```r
## S3 method for class 'diffnet'
c(..., recursive = FALSE)
```

**Arguments**

- `...`  
  diffnet objects to be concatenated.

- `recursive`  
  Ignored.

**Details**

The `diffnet` objects in `...` must fulfill the following conditions:

1. Have the same time range,
2. Have the same vertex attributes, and
3. Have an empty intersection of vertices ids,

The meta data regarding `undirected`, `value`, and `multiple` are set to `TRUE` if any of the concatenating `diffnet` objects has that meta equal to `TRUE`.

The resulting `diffnet` object’s columns in the vertex attributes ordering (both dynamic and static) will coincide with the first `diffnet`’s ordering.

**Value**

A new `diffnet` object with as many vertices as the sum of each concatenated `diffnet` objects’ number of vertices.

**See Also**

Other `diffnet` methods: `%*%`, `as.array.diffnet`, `diffnet-arithmetic`, `diffnet-class`, `diffnet_index`, `plot.diffnet`, `summary.diffnet`
classify_adopters

Examples

# Calculate structural equivalence exposure by city

data(medInnovationsDiffNet)

# Subsetting diffnets

city1 <- medInnovationsDiffNet[medInnovationsDiffNet[["city"]]==1]
city2 <- medInnovationsDiffNet[medInnovationsDiffNet[["city"]]==2]
city3 <- medInnovationsDiffNet[medInnovationsDiffNet[["city"]]==3]
city4 <- medInnovationsDiffNet[medInnovationsDiffNet[["city"]]==4]

# Computing exposure in each one

city1["expo_se"] <- exposure(city1, alt.graph="se", valued=TRUE)
city2["expo_se"] <- exposure(city2, alt.graph="se", valued=TRUE)
city3["expo_se"] <- exposure(city3, alt.graph="se", valued=TRUE)
city4["expo_se"] <- exposure(city4, alt.graph="se", valued=TRUE)

# Concatenating all

diffnet <- c(city1, city2, city3, city4)
diffnet

classify_adopters

Classify adopters accordingly to Time of Adoption and Threshold levels.

Description

Adopters are classified as in Valente (1995). In general, this is done depending on the distance in terms of standard deviations from the mean of Time of Adoption and Threshold.

Usage

classify_adopters(...)

classify(...)

## S3 method for class 'diffnet'
classify_adopters(graph, include_censored = FALSE, ...)

## Default S3 method:
classify_adopters(graph, toa, t0 = NULL, t1 = NULL,
               expo = NULL, include_censored = FALSE, ...)

## S3 method for class 'diffnet_adopters'
ftable(x, as.pcent = TRUE, digits = 2, ...)

## S3 method for class 'diffnet_adopters'

classify_adopters

```r
as.data.frame(x, row.names = NULL,
    optional = FALSE, ...)

## S3 method for class 'diffnet_adopters'
plot(x, y = NULL, ftable.args = list(),
    table.args = list(), ...)

Arguments

... Further arguments passed to the method.
graph A dynamic graph.
include_censored Logical scalar, passed to threshold.
toa Integer vector of length \( n \) with times of adoption.
t0 Integer scalar passed to threshold and toa_mat.
t1 Integer scalar passed to toa_mat.
expo Numeric matrix of size \( n \times T \) with network exposures.
x A diffnet_adopters class object.
as.pcent Logical scalar. When TRUE returns a table with percentages instead.
digits Integer scalar. Passed to round.
row.names Passed to as.data.frame.
only optional Passed to as.data.frame.
y Ignored.
ftable.args List of arguments passed to ftable.
table.args List of arguments passed to table.

Details

Classifies (only) adopters according to time of adoption and threshold as described in Valente (1995). In particular, the categories are defined as follow:

For Time of Adoption, with toa as the vector of times of adoption:

- **Early Adopters**: \( \text{toa}[i] \leq \text{mean}(\text{toa}) - \text{sd}(\text{toa}) \),
- **Early Majority**: \( \text{mean}(\text{toa}) - \text{sd}(\text{toa}) < \text{toa}[i] \leq \text{mean}(\text{toa}) \),
- **Late Majority**: \( \text{mean}(\text{toa}) < \text{toa}[i] \leq \text{mean}(\text{toa}) + \text{sd}(\text{toa}) \), and
- **Laggards**: \( \text{mean}(\text{toa}) + \text{sd}(\text{toa}) < \text{toa}[i] \).

For Threshold levels, with thr as the vector of threshold levels:

- **Very Low Thresh.**: \( \text{thr}[i] \leq \text{mean}(\text{thr}) - \text{sd}(\text{thr}) \),
- **Low Thresh.**: \( \text{mean}(\text{thr}) - \text{sd}(\text{thr}) < \text{thr}[i] \leq \text{mean}(\text{thr}) \),
- **High Thresh.**: \( \text{mean}(\text{thr}) < \text{thr}[i] \leq \text{mean}(\text{thr}) + \text{sd}(\text{thr}) \), and
- **Very High. Thresh.**: \( \text{mean}(\text{thr}) + \text{sd}(\text{thr}) < \text{thr}[i] \).
classify_adopters

By default threshold levels are not computed for left censored data. These will have a NA value in the thr vector.

The plot method, plot.diffnet_adopters, is a wrapper for the plot.table method. This generates a mosaicplot plot.

Value

A list of class diffnet_adopters with the following elements:

- toa: A factor vector of length n with 4 levels: "Early Adopters", "Early Majority", "Late Majority", and "Laggards"
- thr: A factor vector of length n with 4 levels: "Very Low Thresh.", "Low Thresh.", "High Thresh.", and "Very High Thresh."

Author(s)

George G. Vega Yon

References


See Also

Other statistics: bass, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, infection, moran, struct_equiv, threshold, vertex_covariate_dist

Examples

# Classifying brfarmers

```r
x <- brfarmersDiffNet
diffnet.toa(x)[x$toa==max(x$toa, na.rm = TRUE)] <- NA
out <- classify_adopters(x)

# This is one way
round(
  with(out, ftable(toa, thr, dnn=c("Time of Adoption", "Threshold")))/
  nnodes(x[!is.na(x$toa)])*100, digits=2)

# This is other
ftable(out)

# Can be coerced into a data.frame, e.g.
```

```r
## Not run:
View(classify(brfarmersDiffNet))
cbind(as.data.frame(classify(brfarmersDiffNet)), brfarmersDiffNet$toa)

## End(Not run)
```
classify_graph

# Creating a mosaic plot with the medical innovations
-------------
x <- classify(medInnovationsDiffNet)
plot(x)

classify_graph

Analyze an R object to identify the class of graph (if any)

Description
Analyze an R object to identify the class of graph (if any)

Usage
classify_graph(graph)

Arguments

graph Any class of accepted graph format (see netdiffuseR-graphs).

Details
This function analyzes an R object and tries to classify it among the accepted classes in netdiffuseR. If the object fails to fall in one of the types of graphs the function returns with an error indicating what (and when possible, where) the problem lies.

The function was designed to be used with as_diffnet.

Value
Whe the object fits any of the accepted graph formats, a list of attributes including
type Character scalar. Whether is a static or a dynamic graph
class Character scalar. The class of the original object
ids Character vector. Labels of the vertices
pers Integer vector. Labels of the time periods
nper Integer scalar. Number of time periods
n Integer scalar. Number of vertices in the graph

Otherwise returns with error.

Author(s)
George G. Vega Yon

See Also
as_diffnet, netdiffuseR-graphs
cumulative_adopt_count

Cumulative count of adopters

Description
For each time period, calculates the number of adopters, the proportion of adopters, and the adoption rate.

Usage
cumulative_adopt_count(obj)

Arguments
obj A $n \times T$ matrix (Cumulative adoption matrix obtained from toa_mat) or a diffnet object.

Details
The rate of adoption–returned in the 3rd row out the resulting matrix–is calculated as

$$\frac{q_t - q_{t-1}}{q_{t-1}}$$

where $q_t$ is the number of adopters in time $t$. Note that it is only calculated for $t > 1$.

Value
A $3 \times T$ matrix, where its rows contain the number of adopters, the proportion of adopters and the rate of adoption respectively, for each period of time.

Author(s)
George G. Vega Yon & Thomas W. Valente

See Also
Other statistics: bass, classify_adopters, dgr, ego_variance, exposure, hazard_rate, infection, moran, struct_equiv, threshold, vertex_covariate_dist
**Description**

Computes the requested degree measure for each node in the graph.

**Usage**

```r
dgr(graph, cmode = "degree", 
    undirected =getOption("diffnet.undirected", FALSE), 
    self =getOption("diffnet.self", FALSE), 
    valued =getOption("diffnet.valued", FALSE))
```

```r
## S3 method for class 'diffnet_degSeq'
plot(x, breaks = min(100L, nrow(x))/5, 
    freq = FALSE, y = NULL, log = "xy", hist.args = list(), 
    slice = ncol(x), xlab = "Degree", ylab = "Freq", ...)
```

**Arguments**

- `graph` Any class of accepted graph format (see netdiffuserR-graphs).
- `cmode` Character scalar. Either "indegree", "outdegree" or "degree".
- `undirected` Logical scalar. When TRUE only the lower triangle of the adjacency matrix will considered (faster).
- `self` Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
- `valued` Logical scalar. When TRUE weights will be considered. Otherwise non-zero values will be replaced by ones.
- `x` An diffnet_degSeq object
- `breaks` Passed to `hist`.
- `freq` Logical scalar. When TRUE the y-axis will reflex counts, otherwise densities.
- `y` Ignored
- `log` Passed to `plot` (see `par`).
- `hist.args` Arguments passed to `hist`.
- `slice` Integer scalar. In the case of dynamic graphs, number of time point to plot.
- `xlab` Character scalar. Passed to `plot`.
- `ylab` Character scalar. Passed to `plot`.
- `...` Further arguments passed to `plot`.

**Value**

A numeric matrix of size $n \times T$. In the case of plot, returns an object of class `histogram`.
diag_expand

Author(s)

George G. Vega Yon

See Also

Other statistics: bass, classify_adaptors, cumulative_adopt_count, ego_variance, exposure, hazard_rate, infection, moran, struct_equiv, threshold, vertex_covariate_dist

Other visualizations: diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet2, plot_diffnet, plot_infectsuscep, plot_threshold, rescale_vertex_igraph

Examples

```r
# Comparing degree measurements
# Creating an undirected graph
graph <- rgraph.ba()

data.frame(
  In=dgr(graph, "indegree", undirected = FALSE),
  Out=dgr(graph, "outdegree", undirected = FALSE),
  Degree=dgr(graph, "degree", undirected = FALSE)
)

# Testing on Korean Family Planning (weighted graph)
data(kfamilyDiffNet)
d_unvalued <- dgr(kfamilyDiffNet, valued=FALSE)
d_valued <- dgr(kfamilyDiffNet, valued=TRUE)
any(d_valued!=d_unvalued)

# Classic Scale-free plot
set.seed(1122)
g <- rgraph.ba(t=1e3-1)
hist(dgr(g))

# Since by default uses logscale, here we suppress the warnings
# on points been discarded for <=0.
suppressWarnings(plot(dgr(g)))
```

---

diag_expand

Creates a square matrix suitable for spatial statistics models.

Description

Creates a square matrix suitable for spatial statistics models.
Usage

diag_expand(...)

## S3 method for class 'list'
diag_expand(graph, self = getOption("diffnet.self"),
            valued = getOption("diffnet.valued"), ...)

## S3 method for class 'diffnet'
diag_expand(graph, self = getOption("diffnet.self"),
            valued = getOption("diffnet.valued"), ...)

## S3 method for class 'matrix'
diag_expand(graph, nper,
            self = getOption("diffnet.self"),
            valued = getOption("diffnet.valued"), ...)

## S3 method for class 'array'
diag_expand(graph, nper,
            self = getOption("diffnet.self"),
            valued = getOption("diffnet.valued"), ...)

## S3 method for class 'dgCMATRIX'
diag_expand(graph, nper,
            self = getOption("diffnet.self"),
            valued = getOption("diffnet.valued"), ...)

Arguments

... Further arguments to be passed to the method.

graph Any class of accepted graph format (see netdiffuseR\-graphs).

self Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see
details).

valued Logical scalar. When TRUE weights will be considered. Otherwise non-zero
values will be replaced by ones.

nper Integer scalar. Number of time periods of the graph.

Value

A square matrix of class dgCMATRIX of size (\(n_{node}(g)\times n_{per}\))^2

Examples

# Simple example
set.seed(23)
g <- rgraph_er(n=10, p=.5, t=2,undirected=TRUE)

# What we've done: A list with 2 bernoulli graphs
g
# Expanding to a 20*20 matrix with structural zeros on the diagonal
# and on cell 'off' adjacency matrix
diag_expand(g)

---

diffnet-arithmetic  diffnet Arithmetic and Logical Operators

Description

Addition, subtraction, network power of diffnet and logical operators such as & and | as objects

Usage

```r
## S3 method for class 'diffnet'
x ^ y
graph_power(x, y, valued = getOption("diffnet.valued", FALSE))

## S3 method for class 'diffnet'
y / x

## S3 method for class 'diffnet'
x - y

## S3 method for class 'diffnet'
x * y

## S3 method for class 'diffnet'
x & y

## S3 method for class 'diffnet'
x | y
```

Arguments

- `x` A diffnet class object.
- `y` Integer scalar. Power of the network
- `valued` Logical scalar. When FALSE all non-zero entries of the adjacency matrices are set to one.

Details

Using binary operators, ease data management process with diffnet.

By default the binary operator `^` assumes that the graph is valued, hence the power is computed using a weighted edges. Otherwise, if more control is needed, the user can use `graph_power` instead.
Value

A diffnet class object

See Also

Other diffnet methods: `%*%`, `as.array.diffnet`, `c.diffnet`, `diffnet-class`, `diffnet_index`, `plot.diffnet`, `summary.diffnet`

Examples

```r
# Computing two-steps away threshold with the Brazilian farmers data
#--------------------------------
data(brfFarmersDiffNet)

expo1 <- threshold(brfFarmersDiffNet)
expo2 <- threshold(brfFarmersDiffNet^2)

# Computing correlation
cor(expo1, expo2)

# Drawing a qqplot
qqplot(expo1, expo2)

# Working with inverse-----------------------------------------------
brf2_step <- brfFarmersDiffNet^2
brf2_step <- 1/brf2_step

# Removing the first 3 vertex of medInnovationsDiffNet
#-----------------------------------------------
data(medInnovationsDiffNet)

# Using a diffnet object
first3DiffNet <- medInnovationsDiffNet[1:3, ,]
medInnovationsDiffNet - first3DiffNet

# Using indexes
medInnovationsDiffNet - 1:3

# Using ids
medInnovationsDiffNet - as.character(1001:1003)
```

diffnet-class

**Creates a diffnet class object**

Description

diffnet objects contain diffusion networks. With adjacency matrices and time of adoption (toa) vector as its main components, most of the package’s functions have methods for this class of objects.
Usage

as_diffnet(graph, ...)

## Default S3 method:
as_diffnet(graph, ...)

## S3 method for class 'networkDynamic'
as_diffnet(graph, toavar, ...)

new_diffnet(graph, toa, t0 = min(toa, na.rm = TRUE), t1 = max(toa, na.rm = TRUE),
vertex.dyn.attrs = NULL, vertex.static.attrs = NULL,
id.and.per.vars = NULL, graph.attrs = NULL,
undirected = getOption("diffnet.undirected"),
self = getOption("diffnet.self"),
multiple = getOption("diffnet.multiple"), name = "Diffusion Network",
behavior = "Unspecified")

## S3 method for class 'diffnet'
as.data.frame(x, row.names = NULL, optional = FALSE,
attr.class = c("dyn", "static"), ...)

diffnet.attrs(graph, element = c("vertex", "graph"),
attr.class = c("dyn", "static"), as.df = FALSE)

diffnet.toa(graph)
diffnet.toa(graph, i) <- value

## S3 method for class 'diffnet'
print(x, ...)

nodes(graph)
diffnetLapply(graph, FUN, ...)

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

## S3 method for class 'diffnet'
dimnames(x)

## S3 method for class 'diffnet'
t(x)

## S3 method for class 'diffnet'
Arguments

- **graph**: A dynamic graph (see `netdiffuseR-graphs`).
- **...**: Further arguments passed to the jmethod.
- **toavar**: Character scalar. Name of the variable that holds the time of adoption.
- **toa**: Numeric vector of size $n$. Times of adoption.
- **t0**: Integer scalar. Passed to `toa_mat`.
- **t1**: Integer scalar. Passed to `toa_mat`.
- **vertex.dyn.attrs**: Vertices dynamic attributes (see details).
- **vertex.static.attrs**: Vertices static attributes (see details).
- **id.and.per vars**: A character vector of length 2. Optionally specified to check the order of the rows in the attribute data.
- **graph.attrs**: Graph dynamic attributes (not supported yet).
- **undirected**: Logical scalar. When TRUE only the lower triangle of the adjacency matrix will be considered (faster).
- **self**: Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
- **multiple**: Logical scalar. When TRUE allows multiple edges.
- **name**: Character scalar. Name of the diffusion network (descriptive).
- **behavior**: Character scalar. Name of the behavior been analyzed (innovation).
- **x**: A `diffnet` object.
- **row.names**: Ignored.
- **optional**: Ignored.
- **attr.class**: Character vector/scalar. Indicates the class of the attribute, either dynamic ("dyn"), or static ("static").
- **element**: Character vector/scalar. Indicates what to retrieve/alter.
- **as.df**: Logical scalar. When TRUE returns a data.frame.
- **value**: In the case of `diffnet.toa`, replacement, otherwise see below.
- **i**: Indices specifying elements to replace. See `Extract`.
- **FUN**: a function to be passed to `lapply`
- **object**: A `diffnet` object.

Details

diffnet objects hold both, static and dynamic vertex attributes. When creating `diffnet` objects, these can be specified using the arguments `vertex.static.attrs` and `vertex.dyn.attrs`; depending on whether the attributes to specify are static or dynamic, `netdiffuseR` currently supports the following objects:
The last column, **Check sorting**, lists the variables that the user should specify if he wants the function to check the order of the rows of the attributes (notice that this is not possible for the case of vectors). By providing the name of the vertex id variable, `id`, and the time period id variable, `per`, the function makes sure that the attribute data is presented in the right order. See the example below. If the user does not provide the names of the vertex id and time period variables then the function does not check the way the rows are sorted, further it assumes that the data is in the correct order.

**Value**

A list of class `diffnet` with the following elements:

- **graph**: A list of length `T`. Containing sparse square matrices of size `n` and class `dgCMatrix`.
- **toa**, **cumadopt**: An integer vector of size `T` with times of adoption.
- **adoptL**, **cumadoptL**: Numeric matrices of size `n x T` as those returned by `toa_mat`.
- **vertexNstaticNattrs**: If not NULL, a data frame with `n` rows with vertex static attributes.
- **vertexNdynNattrs**: A list of length `T` with data frames containing vertex attributes throughout time (dynamic).
- **graphNattrs**: A data frame with `T` rows.
- **meta**: A list of length 9 with the following elements:
  - **type**: Character scalar equal to "dynamic".
  - **class**: Character scalar equal to "list".
  - **ids**: Character vector of size `n` with vertices’ labels.
  - **pers**: Integer vector of size `T`.
  - **nper**: Integer scalar equal to `T`.
  - **n**: Integer scalar equal to `n`.
  - **self**: Logical scalar.
  - **undirected**: Logical scalar.
  - **multiple**: Logical scalar.
  - **name**: Character scalar.
  - **behavior**: Character scalar.
**Auxiliary functions**

`diffnet.attrs` Allows retrieving network attributes. In particular, by default returns a list of length $T$ with data frames with the following columns:

1. `per` Indicating the time period to which the observation corresponds.
2. `toa` Indicating the time of adoption of the vertex.
3. Further columns depending on the vertex and graph attributes.

Each vertex static attributes’ are repeated $T$ times in total so that these can be binded (`rbind`) to dynamic attributes.

When `as.df=TRUE`, this convenience function is useful as it can be used to create event history (panel data) datasets used for model fitting.

Conversely, the replacement method allows including new vertex or graph attributes either dynamic or static (see examples below).

`diffnet.toa(graph)` works as an alias of `graph$toa`. The replacement method, `diffnet.toa<-` used as `diffnet.toa(graph)<-`, is the right way of modifying times of adoption as when doing so it performs several checks on the time ranges, and recalculates adoption and cumulative adoption matrices using `toa_mat`.

`nodes(graph)` is an alias for `graph$meta$ids`.

**Author(s)**

George G. Vega Yon

**See Also**

Default options are listed at `netdiffuseR-options`

Other `diffnet` methods: `%%`, `as.array.diffnet`, `c.diffnet`, `diffnet-arithmetic`, `diffnet_index`, `plot.diffnet`, `summary.diffnet`

Other data management functions: `edgelist_to_adjmat`, `egonet_attrs`, `isolated`, `survey_to_diffnet`

**Examples**

```r
# Creating a random graph
set.seed(123)
graph <- rgraph.ba(t=9)
graph <- lapply(1:5, function(x) graph)

# Pretty TOA
names(graph) <- 2001L:2005L
toa <- sample(c(2001L:2005L,NA), 10, TRUE)

diffnet <- new_diffnet(graph, toa)
diffnet
summary(diffnet)
```

# Plotting slice 4
plot(diffnet, t=4)

# ATTRIBUTES

# Retrieving attributes
diffnet.attrs(diffnet, "vertex", "static")

# Now as a data.frame (only static)
diffnet.attrs(diffnet, "vertex", "static", as.df = TRUE)

# Now as a data.frame (all of them)
diffnet.attrs(diffnet, as.df = TRUE)
as.data.frame(diffnet) # This is a wrapper

# Unsorted data

# Loading example data
data(fakesurveyDyn)

data(fakesurveyDyn, "id", c("net1", "net2", "net3"), "toa", "group", timevar = "time", keep.isolates=TRUE, warn.coercion=FALSE)

# Now, we extract the graph data and create a diffnet object from scratch

# Creating a diffnet object
fs_diffnet <- survey_to_diffnet(
  fakesurveyDyn, "id", c("net1", "net2", "net3"), "toa", "group", timevar = "time", keep.isolates=TRUE, warn.coercion=FALSE)

# Now, we extract the graph data and create a diffnet object from scratch
graph <- fs_diffnet$graph
ids <- fs_diffnet$meta$ids

g <- Map(function(g) {
  dimnames(g) <- list(ids,ids)
  g
  }, g=graph)

# Lets apply a different sorting to the data to see if it works

n <- nrow(attr)
attrs <- attr[order(runif(n)),]

# Now, recreating the old diffnet object (notice -id.and.per.vars- arg)

# Now, retrieving attributes. The 'new one' will have more (repeated)

# Comparing elements!
tocompare <- intersect(colnames(attr_new), colnames(attr_old))
all(attr_new[, tocompare] == attr_old[, tocompare], na.rm = TRUE) # TRUE!

# diffnetLapply

data(medInnovationsDiffNet)
```
diffnetLapply(medInnovationsDiffNet, function(x, cumadopt, ...) {sum(cumadopt)})

diffnet_check_attr_class

Infer whether value is dynamic or static.

Description

Intended for internal use only, this function is used in `diffnet_index` methods.

Usage

diffnet_check_attr_class(value, meta)

Arguments

value
Either a matrix, data frame or a list. Attribute values.

meta
A list. A diffnet object’s meta data.

Value

The value object either as a data frame (if static) or as a list of data frames (if dynamic). If `value` does not follows the permitted types of `diffnet_index`, then returns with error.

```
diffnet_index

Indexing diffnet objects (on development)

Description

Access and assign (replace) elements from the adjacency matrices or the vertex attributes data frames.

Usage

```R
## S3 method for class 'diffnet'
x[[name, as.df = FALSE]]

## S3 replacement method for class 'diffnet'
x[[i, j]] <- value

## S3 method for class 'diffnet'
x[i, j, k, drop = FALSE]

## S3 replacement method for class 'diffnet'
x[i, j, k] <- value
```
Arguments

- **x**: A diffnet class object.
- **name**: String vector. Names of the vertices attributes.
- **as.df**: Logical scalar. When TRUE returns a data frame, otherwise a list of length T.
- **i**: Index of the i-th row of the adjacency matrix (see details).
- **j**: Index of the j-th column of the adjacency matrix (see details).
- **value**: Value to assign (see details).
- **k**: Index of the k-th slice of the adjacency matrix (see details).
- **drop**: Logical scalar. When TRUE returns an adjacency matrix, otherwise a filtered diffnet object.
- **...**: Further arguments to be passed to the method (on development)

Details

The [diffnet] methods provides access to the diffnet attributes data frames, static and dynamic. By providing the name of the corresponding attribute, depending on whether it is static or dynamic the function will return either a data frame–static attributes–or a list of these–dynamic attributes. For the assigning method, [[<-.diffnet, the function will infer what kind of attribute is by analyzing the dimensions of value, in particular we have the following possible cases:

<table>
<thead>
<tr>
<th>Class</th>
<th>Dimension</th>
<th>Inferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix</td>
<td>$n \times T$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>matrix</td>
<td>$n \times 1$</td>
<td>Static</td>
</tr>
<tr>
<td>matrix</td>
<td>$(n \times T) \times 1$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>data.frame</td>
<td>$n \times T$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>data.frame</td>
<td>$n \times 1$</td>
<td>Static</td>
</tr>
<tr>
<td>data.frame</td>
<td>$(n \times T) \times 1$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>vector</td>
<td>$n$</td>
<td>Static</td>
</tr>
<tr>
<td>vector</td>
<td>$n \times T$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>list*</td>
<td>$T$ data.frames/matrices/vectors</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

*: With $n \times 1$ data.frame/matrix or $n$ length vector.

Other cases will return with error.

In the case of the slices index k, either an integer vector with the positions, a character vector with the labels of the time periods or a logical vector of length T can be used to specify which slices to retrieve. Likewise, indexing vertices works in the same way with the only difference that, instead of time period labels and a logical vector of length T, vertices ids labels and a logical vector of length n should be provided.

When subsetting slices, the function modifies the toa vector as well as the adopt and cumadopt matrices collapsing network timing. For example, if a network goes from time 1 to 20 and we set $k=3:10$, all individuals who adopted prior to time 3 will be set as adopters at time 3, and all individuals who adopted after time 10 will be set as adopters at time 10, changing the adoption and cumulative adoption matrices. Importantly, k have no gaps, and it should be within the graph time period range.
Value

In the case of the assigning methods, a diffnet object. Otherwise, for `[.diffnet` a vector extracted from one of the attributes data frames, and for `[.diffnet` a list of length `length(k)` with the corresponding `[i,j]` elements from the adjacency matrix.

Author(s)

George G. Vega Yon

See Also

Other diffnet methods: `%>%` , `as.array.diffnet`, `c.diffnet`, `diffnet-arithmetic`, `diffnet-class`, `plot.diffnet`, `summary.diffnet`

Examples

```r
# Creating a random diffusion network ---------------------------------------------
set.seed(111)
graph <- rdiffrnet(100, 5)

# Accessing to a static attribute
graph["real_threshold"]

# Accessing to subsets of the adjacency matrix
graph[1, 1:3, drop=TRUE]
graph[, 1:3, drop=TRUE][[1]]

# ... Now, as diffnet objects (the default)
graph[1, 1:3, drop=FALSE]

# Changing values in the adjacency matrix
graph[1, , drop=TRUE]
graph[, ,] <- -5

# Adding attributes (dynamic) -----------------------------------------------------
# Preparing the data
set.seed(1122)
x <- rdiffrnet(30, 5, seed.p.adopt=.15)

# Calculating exposure, and storing it differ
expoM <- exposure(x)
expoL <- lapply(seq_len(x$meta$nper), function(x) expoM[, x, drop=FALSE])
expoD <- do.call(rbind, expoL)

# Adding data (all these are equivalent)
x[["expoM"]]<- expoM
x[["expoL"]]<- expoL
x[["expoD"]]<- expoD
```
# Lets compare

identical(x[['expoM']], x[['expoL']]) # TRUE
identical(x[['expoM']], x[['expoD']]) # TRUE

**diffreg**

**Diffusion regression model**

**Description**

A wrapper of glm, this function estimates a lagged regression model of adoption as a function of exposure and other controls as specified by the user.

**Usage**

diffreg(model, type = c("logit", "probit"))

**Arguments**

- **model**: An object of class formula where the right-hand-side is an object of class diffnet
- **type**: Character scalar. Either "probit" or "logit".

**Details**

The model must be in the following form:

<diffnet object> ~ exposure + covariate1 + covariate2 + ...

Where exposure can be specified either as a simple term, or as a call to the exposure function, e.g. to compute exposure with a lag of length 2, the formula could be:

<diffnet object> ~ exposure(lags = 2) + covariate1 + covariate2 + ...

When no argument is passed to exposure, the function sets a lag of length 1 by default (see the Lagged regression section).

This is a wrapper of glm. The function does the following steps:

1. Compute exposure by calling exposure on the LHS.
2. Modify the formula so that the model is on adoption as a function of exposure and whatever covariates the user specifies.
3. Selects either "probit" or "logit" and prepares the call to glm. This includes passing the following line:

   
   subset = ifelse(is.na(toa), TRUE, toa >= per)

   
   This results in including observations that either did not adopt or up to the time of adoption.
4. Estimates the model.

The data passed to glm is obtained by using as.data.frame.diffnet.
**Value**

An object of class `glm`.

**Lagged regression**

The model estimated is a lagged regression model that has two main assumptions:

1. The network is exogenous to the behavior (no selection effect)
2. The influence effect (diffusion) happens in a lagged fashion, hence, exposure is computed lagged.

If either of these two assumptions is not met, then the model becomes endogenous, and so inference becomes invalid.

In the case of the first assumption, the user can overcome the non-exogeneity problem by providing an alternative network. This can be done by specifying `alt.graph` in the `exposure` function so that the network becomes exogenous to the adoption.

**Examples**

```r
# Load data
data("medInnovationsDiffNet")

# Default model
ans <- diffreg(
  medInnovationsDiffNet ~ exposure + factor(city) + proage + per)
summary(ans)
```

**diffusion-data**

**Diffusion Network Datasets**

**Description**

Diffusion Network Datasets

**Details**

The three classic network diffusion datasets included in netdiffuseR are the medical innovation data originally collected by Coleman, Katz & Menzel (1966); the Brazilian Farmers collected as part of the three country study implemented by Everett Rogers (Rogers, Ascroft, & Röling, 1970), and Korean Family Planning data collected by researchers at the Seoul National University’s School of Public (Rogers & Kincaid, 1981). The table below summarizes the three datasets:

<table>
<thead>
<tr>
<th>Country</th>
<th>Medical Innovation</th>
<th>Brazilian Farmers</th>
<th>Korean Family Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Country</strong></td>
<td>USA</td>
<td>Brazil</td>
<td>Korean</td>
</tr>
<tr>
<td><strong># Respondents</strong></td>
<td>125 Doctors</td>
<td>692 Farmers</td>
<td>1,047 Women</td>
</tr>
<tr>
<td><strong># Communities</strong></td>
<td>4</td>
<td>11</td>
<td>25</td>
</tr>
<tr>
<td><strong>Innovation</strong></td>
<td>Tetracycline</td>
<td>Hybrid Corn Seed</td>
<td>Family Planning</td>
</tr>
<tr>
<td><strong>Time for Diffusion</strong></td>
<td>18 Months</td>
<td>20 Years</td>
<td>11 Years</td>
</tr>
<tr>
<td><strong>Year Data Collected</strong></td>
<td>1955-1956</td>
<td>1966</td>
<td>1973</td>
</tr>
</tbody>
</table>
### References

diffusionMap

Description

Using bi-dimensional kernel smoothers, creates a heatmap based on a graph layout and colored accordingly to \( x \). This visualization technique is intended to be used with large graphs.

Usage

diffusionMap(graph, ...)

diffmap(graph, ...)

## Default S3 method:
diffusionMap(graph, x, x.adj = round_to_seq,
layout = NULL, jitter.args = list(), kde2d.args = list(n = 100),
sharp.criter = function(x, w) {
  wvar(x, w) > (max(x, na.rm = TRUE)
- min(x, na.rm = TRUE))\(^2/12\}, ...)

## S3 method for class 'diffnet'
diffusionMap(graph, slice = nslices(graph), ...)

## S3 method for class 'diffnet_diffmap'
image(x, ...)

## S3 method for class 'diffnet_diffmap'
print(x, ...)

## S3 method for class 'diffnet_diffmap'
plot(x, y = NULL, ...)

Arguments

- **graph**: A square matrix of size \( n \times n \).
- **...**: Arguments passed to method.
- **x**: An vector of length \( n \). Usually a toa vector.
- **x.adj**: Function to adjust \( x \). If not NULL then it is applied to \( x \) at the beginning (see details).
- **layout**: Either a \( n \times 2 \) matrix of coordinates or a layout function applied to graph (must return coordinates).
diffusionMap

jitter.args  A list including arguments to be passed to jitter.
kde2d.args  A list including arguments to be passed to kde2d.
sharp.criter A function choose whether to apply a weighted mean for each cell, or randomize over the values present in that cell (see details).
slice  Integer scalar. Slice of the network to be used as baseline for drawing the graph.
y  Ignored.

Details

The image is created using the function kde2d from the MASS package. The complete algorithm follows:

1. x is coerced into integer and the range is adjusted to start from 1. NA are replaced by zero.
2. If no layout is passed, layout is computed using layout_nicely from igraph
3. Then, a kde2d map is computed for each level of x. The resulting matrices are added up as a weighted sum. This only holds if at the cell level the function sharp.criter returns FALSE.
4. The jitter function is applied to the repeated coordinates.
5. 2D kernel is computed using kde2d over the coordinates.

The function sharp.criter must take two values, a vector of levels and a vector of weights. It must return a logical scalar with value equal to TRUE when a randomization at the cell level must be done, in which case the final value of the cell is chosen using sample(x, 1, prob=w).

The resulting matrix can be passed to image or similar.

The argument x.adj uses by default the function round_to_seq which basically maps x to a fix length sequence of numbers such that x.adj(x) resembles an integer sequence.

Value

A list of class diffnet_diffmap

coords  A matrix of size n × 2 of vertices coordinates.
map  Output from kde2d. This is a list with 3 elements, vectors x, y and matrix z of size n × n (passed via kde2d.args).
h  Bandwidth passed to kde2d.

Author(s)

George G. Vega Yon

References

Vega Yon, George G., and Valente, Thomas W., Visualizing Large Annotated Networks as Heatmaps using Weighted Averages based on Kernel Smoothers (Working paper).

See Also

Other visualizations: dgr, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet2, plot_diffnet, plot_infectsuscep, plot_threshold, rescale_vertex_igraph
Examples

```r
# Example with a random graph -----------------------------------------------

## Not run:
set.seed(1231)

# Random scale-free diffusion network
x <- rdiffnet(1000, 4, seed.graph="scale-free", seed.p.adopt = .025,
    rewire = FALSE, seed.nodes = "central",
    rgraph.arg=list(self=FALSE, m=4),
    threshold.dist = function(id) runif(1,.2,.4))

# Diffusion map (no random toa)
dm0 <- diffusionMap(x, kde2d.args=list(n=150, h=.5), layout=igraph::layout_with_fr)

# Random
diffnet.toa(x) <- sample(x$toa, size = nnodes(x))

# Diffusion map (random toa)
dm1 <- diffusionMap(x, layout = dm0$coords, kde2d.args=list(n=150, h=.5))

oldpar <- par(no.readonly = TRUE)
col <- colorRampPalette(blues9)(100)
par(mfrow=c(1,2), oma=c(1,0,0,0))
image(dm0, col=col, main="Non-random Times of Adoption\nAdoption from the core.")
image(dm1, col=col, main="Random Times of Adoption")
par(mfrow=c(1,1))
mtext("Both networks have the same distribution on times of adoption", 1,
    outer = TRUE)
par(oldpar)

## End(Not run)

# Example with Brazilian Farmers -------------------------------------------

## Not run:
dn <- brfarmersDiffNet

# Setting last TOA as NA
diffnet.toa(dn)[dn$toa == max(dn$toa)] <- NA

# Coordinates
coords <- sna::gplot.layout.fruchtermanreingold(
    as.matrix(dn$graph[[1]]), layout.par=NULL
)

# Plotting diffusion
plot_diffnet2(dn, layout=coords, vertex.size = 300)

# Adding diffusion map
out <- diffusionMap(dn, layout=coords, kde2d.args=list(n=100, h=50))
```
drawColorKey

```r
col <- adjustcolor(colorRampPalette(c("white","lightblue","yellow","red"))(100), .5)
with(out$map, .filled.contour(x,y,z,pretty(range(z), 100),col))

## End(Not run)
```

drawColorKey  

**Draw a color key in the current device**

**Description**

Draw a color key in the current device

**Usage**

```r
drawColorKey(x, tick.marks = pretty_within(x), labels = tick.marks,
main = NULL, key.pos = c(0.925, 0.975, 0.05, 0.95), pos = 2,
nlevels = length(tick.marks),
color.palette = viridisLite::viridis(nlevels), tick.width = c(0.01,
0.0075), add.box = TRUE, na.col = NULL, na.height = 0.1,
na.lab = "n/a", ...)
```

**Arguments**

- **x**: A numeric vector with the data (it is used to extract the range).
- **tick.marks**: A numeric vector indicating the levels to be included in the axis.
- **labels**: Character vector. When provided, specifies using different labels for the tick marks than those provided by `tick.marks`.
- **main**: Character scalar. Title of the key.
- **key.pos**: A numeric vector of length 4 with relative coordinates of the key (as % of the plotting area, see `par("usr")`)
- **pos**: Integer scalar. Position of the axis as in `text`.
- **nlevels**: Integer scalar. Number of levels (colors) to include in the color key.
- **color.palette**: Color palette of length(nlevels).
- **tick.width**: Numeric vector of length 2 indicating the length of the inner and outer tick marks as percentage of the axis.
- **add.box**: Logical scalar. When TRUE adds a box around the key.
- **na.col**: Character scalar. If specified, adds an additional box indicating the NA color.
- **na.height**: Numeric scalar. Relative height of the NA box. Only use if `na.col` is not NULL.
- **na.lab**: Character scalar. Label of the NA block. Only use if `na.col` is not NULL.
- **...**: Further arguments to be passed to `rect`
edgelist_to_adjmat

Value
Invisible NULL.

Author(s)
George G. Vega Yon

See Also
Other visualizations: dgr, diffusionMap, grid_distribution, hazard_rate, plot_adopters,
plot_diffnet2, plot_diffnet, plot_infectsuscep, plot_threshold, rescale_vertex_igraph

Examples
set.seed(166)
x <- rnorm(100)
col <- colorRamp(c("lightblue", "yellow", "red")(x - min(x))/(max(x) - min(x)))
col <- rgb(col, maxColorValue = 255)
plot(x, col=col, pch=19)
drawColorKey(x, nlevels = 100, border="transparent",
main="Key\n\nLike A\nBoss")

edgelist_to_adjmat Conversion between adjacency matrix and edgelist

Description
Generates adjacency matrix from an edgelist and vice versa.

Usage
edgelist_to_adjmat(edgelist, w = NULL, t0 = NULL, t1 = NULL,
t = NULL, simplify = TRUE,
undirected = getOption("diffnet.undirected"),
self = getOption("diffnet.self"),
multiple = getOption("diffnet.multiple"), keep.isolates = TRUE,
recode.ids = TRUE)

adjmat_to_edgelist(graph, undirected = getOption("diffnet.undirected",
FALSE), keep.isolates = getOption("diffnet.keep.isolates", TRUE))

Arguments
edgelist Two column matrix/data.frame in the form of ego-source- and alter-target- (see
details).
w Numeric vector. Strength of ties (optional).
t0 Integer vector. Starting time of the ties (optional).
edgelist_to_adjmat

- **t1**: Integer vector. Finishing time of the ties (optional).
- **t**: Integer scalar. Repeat the network t times (if no t0, t1 are provided).
- **simplify**: Logical scalar. When TRUE and times=NULL it will return an adjacency matrix, otherwise an array of adjacency matrices. (see details).
- **undirected**: Logical scalar. When TRUE only the lower triangle of the adjacency matrix will be considered (faster).
- **self**: Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
- **multiple**: Logical scalar. When TRUE allows multiple edges.
- **keep.isolates**: Logical scalar. When FALSE, rows with NA/NULL values (isolated vertices unless have autolink) will be dropped (see details).
- **recode.ids**: Logical scalar. When TRUE ids are recoded using as.factor (see details).
- **graph**: Any class of accepted graph format (see netdiffuserR-graphs).

**Details**

When converting from edgelist to adjmat the function will recode the edgelist before starting. The user can keep track after the recording by checking the resulting adjacency matrices' row.names. In the case that the user decides skipping the recoding (because wants to keep vertices index numbers, implying that the resulting graph will have isolated vertices), he can override this by setting recode.ids=FALSE (see example).

When multiple edges are included, multiple=TRUE, each vertex between \{i,j\} will be counted as many times it appears in the edgelist. So if a vertex \{i,j\} appears 2 times, the adjacency matrix element \{i,j\} will be 2.

Edges with incomplete information (missing data on w or times) are not included on the graph. Incomplete cases are tagged using complete.cases and can be retrieved by the user by accessing the attribute incomplete.

Were the case that either ego or alter are missing (i.e. NA values), the function will either way include the non-missing vertex. See below for an example of this.

The function performs several checks before starting to create the adjacency matrix. These are:

- Dimensions of the inputs, such as number of columns and length of vectors
- Having complete cases. If any edge has a non-numeric value such as NAs or NULL in either times or w, it will be removed. A full list of such edges can be retrieved from the attribute incomplete
- Nodes and times ids coding

recode.ids=FALSE is useful when the vertices ids have already been coded. For example, after having use adjmat_to_edgelist, ids are correctly encoded, so when going back (using edgelist_to_adjmat) recode.ids should be FALSE.

**Value**

In the case of edgelist_to_adjmat either an adjacency matrix (if times is NULL) or an array of these (if times is not null). For adjmat_to_edgelist the output is an edgelist with the following columns:
edgelist_to_adjmat

- `ego`: Origin of the tie.
- `alter`: Target of the tie.
- `value`: Value in the adjacency matrix.
- `time`: Either a 1 (if the network is static) or the time stamp of the tie.

**Author(s)**

George G. Vega Yon & Thomas W. Valente

**See Also**

Other data management functions: `diffnet-class, egonet.attrs, isolated, survey_to_diffnet`

**Examples**

```r
# Base data
set.seed(123)
n <- 5
edgelist <- rgraph_er(n, as.edgelist=TRUE, p=.2)[,c("ego","alter")]
times <- sample.int(3, nrow(edgelist), replace=TRUE)
w <- abs(rnorm(nrow(edgelist)))

# Simple example
edgelist_to_adjmat(edgelist)
edgelist_to_adjmat(edgelist, undirected = TRUE)

# Using w
edgelist_to_adjmat(edgelist, w)
edgelist_to_adjmat(edgelist, w, undirected = TRUE)

# Using times
edgelist_to_adjmat(edgelist, t0 = times)
edgelist_to_adjmat(edgelist, t0 = times, undirected = TRUE)

# Using times and w
edgelist_to_adjmat(edgelist, t0 = times, w = w)
edgelist_to_adjmat(edgelist, t0 = times, undirected = TRUE, w = w)

# Not recoding  -------------------------------
# Notice that vertices 3, 4 and 5 are not present in this graph.
graph <- matrix(c(
1,2,6,
6,6,7
), ncol=2)

# Generates an adjmat of size 4 x 4
edgelist_to_adjmat(graph)

# Generates an adjmat of size 7 x 7
edgelist_to_adjmat(graph, recode.ids=FALSE)
```
Compute ego/alter edge coordinates considering alter's size and aspect ratio

Description

Given a graph, vertices' positions and sizes, calculates the absolute positions of the endpoints of the edges considering the plot's aspect ratio.

Usage

edges_coords(graph, toa, x, y, vertex_cex, undirected = TRUE, no_contemporary = TRUE, dev = as.numeric(c()), ran = as.numeric(c()), curved = as.logical(c()))

Arguments

- graph: A square matrix of size n. Adjacency matrix.
- toa: Integer vector of size n. Times of adoption.
- x: Numeric vector of size n. x-coordinate of vertices.
- y: Numeric vector of size n. y-coordinate of vertices.
vertex_cex Numeric vector of size n. Vertices’ sizes in terms of the x-axis (see symbols).

undirected Logical scalar. Whether the graph is undirected or not.

no_contemporary Logical scalar. Whether to return (compute) edges’ coordinates for vertices with the same time of adoption (see details).

dev Numeric vector of size 2. Height and width of the device (see details).

ran Numeric vector of size 2. Range of the x and y axis (see details).

curved Logical vector.

Details

In order to make the plot’s visualization more appealing, this function provides a straightforward way of computing the tips of the edges considering the aspect ratio of the axes range. In particular, the following corrections are made at the moment of calculating the edges coords:

- Instead of using the actual distance between ego and alter, a relative one is calculated as follows

\[ d' = \left( (x_0 - x_1)^2 + (y'_0 - y'_1)^2 \right)^{\frac{1}{2}} \]

where \( y'_i = y_i \times \frac{\max y - \min y}{\max y - \min y} \)

- Then, for the relative elevation angle, \( \alpha' \), the relative distance \( d' \) is used, \( \alpha' = \arccos\left(\frac{(x_0 - x_1)}{d'}\right) \)

- Finally, the edge’s endpoint’s (alter) coordinates are computed as follows:

\[ x'_1 = x_1 + \cos(\alpha') \times v_1 \]
\[ y'_1 = y_1 - \sin(\alpha') \times v_1 \times \frac{\max y - \min y}{\max x - \min x} \]

Where \( v_1 \) is alter’s size in terms of the x-axis, and the sign of the second term in \( y'_1 \) is negative iff \( y_0 < y_1 \).

The same process (with sign inverted) is applied to the edge starting point. The resulting values, \( x'_1, y'_1 \) can be used with the function \( \text{arrows} \). This is the workhorse function used in \( \text{plot_threshold} \).

The \( \text{dev} \) argument provides a reference to rescale the plot accordingly to the device, and former, considering the size of the margins as well (this can be easily fetched via \( \text{par("pin")} \), plot area in inches).

On the other hand, \( \text{ran} \) provides a reference for the adjustment according to the range of the data, this is \( \text{range(x)[2]} - \text{range(x)[1]} \) and \( \text{range(y)[2]} - \text{range(y)[1]} \) respectively.

Value

A numeric matrix of size \( m \times 5 \) with the following columns:

- \( x_0, y_0 \) Edge origin
- \( x_1, y_1 \) Edge target
- \( \alpha' \) Relative angle between \( (x_0, y_0) \) and \( (x_1, y_1) \) in terms of radians

With \( m \) as the number of resulting edges.
**Examples**

```r
# data(medInnovationsDiffNet)
library(sna)

data(medInnovationsDiffNet)
library(sna)

# Computing coordinates
set.seed(79)
coors <- sna::gplot(as.matrix(medInnovationsDiffNet$graph[[1]]))

# Getting edge coordinates
vcex <- rep(1.5, nnodes(medInnovationsDiffNet))
ecoords <- edges_coords(
  medInnovationsDiffNet$graph[[1]],
  diffnet.toa(medInnovationsDiffNet),
  x = coors[,1], y = coors[,2],
  vertex_cex = vcex,
  dev = par("pin")
)

ecoords <- as.data.frame(ecoords)

# Plotting
symbols(ecoords[,1], coors[,2], circles=vcex,
  inches=FALSE, xaxs="i", yaxs="i")
with(ecoords, arrows(x0,y0,x1,y1, length=.1))
```

---

**egonet_attrs**

*Retrieve alter’s attributes (network effects)*

**Description**

For a given set of vertices V, retrieves each vertex’s alter’s attributes. This function enables users to calculate exposure on variables other than the attribute that is diffusing. Further, it enables the specification of alternative functions to use to characterize ego’s personal network including calculating the mean, maximum, minimum, median, or sum of the alters’ attributes. These measures may be static or dynamic over the interval of diffusion and they may be binary or valued.

**Usage**

```r
egonet_attrs(graph, attrs, V = NULL, direction = "outgoing",
  fun = function(x) x, as.df = FALSE,
  self =getOption("diffnet.self"),
  valued =getOption("diffnet.valued"), ...)
```

**Arguments**

- **graph**: Any class of accepted graph format (see `netdiffuseR-graphs`).
egonet_attrs

attrs  If graph is static, Numeric matrix with \( n \) rows, otherwise a list of numeric matrices with \( n \) rows.

\[ v \] Integer vector. Set of vertices from which the attributes will be retrieved.

direction Character scalar. Either "outgoing", "incoming".

fun Function. Applied to each

as.df Logical scalar. When TRUE returns a data.frame instead of a list (see details).

self Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).

valued Logical scalar. When TRUE weights will be considered. Otherwise non-zero values will be replaced by ones.

... Further arguments to be passed to fun.

Details

By indexing inner/outer edges, this function retrieves ego network attributes for all \( v \in V \), which by default is the complete set of vertices in the graph.

When as.df=TRUE the function returns a data.frame of size \( (|V| \times T) \times k \) where \( T \) is the number of time periods and \( k \) is the number of columns generated by the function.

The function can be used to create network effects as those in the \textbf{RSiena} package. The difference here is that the definition of the statistic directly relies on the user. For example, in the \textbf{RSiena} package, the dyadic covariate effect 37. covariate (centered) main effect \( (X) \)

\[ s_{i37}(x) = \sum_j x_{ij}(w_{ij} - \bar{w}) \]

Which, having a diffnet object with attributes named \( x \) and \( w \), can be calculated as

```r
egonet_attrs(diffnet, as.df=TRUE, fun=function(dat) {
  sum(dat[, "x"]*(dat[, "w"] - mean(dat[, "w"])))
})
```

Furthermore, we could use the \textit{median} centered instead, for example

```r
egonet_attrs(diffnet, as.df=TRUE, fun=function(dat) {
  sum(dat[, "x"]*(dat[, "w"] - median(dat[, "w"])))
})
```

Where for each \( i \), dat will be a matrix with as many rows as individuals in his egonetwork. Such matrix holds the column names of the attributes in the network.

When \texttt{self = TRUE}, it will include ego’s attributes, regardless the network has loops or not.
Value

A list with ego alters’s attributes. By default, if the graph is static, the output is a list of length length(V) with matrices having the following columns:

- **value**: Either the corresponding value of the tie.
- **id**: Alter’s id
- **...**: Further attributes contained in `attrs`

On the other hand, if graph is dynamic, the output is list of length \( T \) of lists of length \( \text{length}(V) \) with data frames having the following columns:

- **value**: The corresponding value of the adjacency matrix.
- **id**: Alter’s id
- **per**: Time id
- **...**: Further attributes contained in `attrs`

Author(s)

George G. Vega Yon

See Also

Other data management functions: `diffnet-class`, `edgelist_to_adjmat`, `isolated`, `survey_to_diffnet`

Examples

```r
# Simple example with diffnet
set.seed(1001)
diffnet <- rdiffnet(150, 5, seed.graph="small-world")

# Adding attributes
indeg <- dgr(diffnet, cmode="indegree")
head(indeg)
diffnet[['indegree']] <- indeg

# Retrieving egonet’s attributes (vertices 1 and 20)
egonetAttrs(diffnet, V=c(1,20))

# Example with a static network
set.seed(1231)
n <- 20
net <- rgraph_ws(n = n, k = 4, p = .5)
someattr <- matrix(rnorm(n * 2), ncol = 2, dimnames = list(NULL, c("a", "b")))

# Maximum of -a- in ego network
ans <- egonet_attrs(net, someattr, fun = function(x) max(x[,"a"]))
ans

# checking it worked, taking a look at node 1, 2, and 3
```
ego_variance  Computes variance of Y at ego level

Description

Computes variance of Y at ego level

Usage

ego_variance(graph, Y, funname, all = FALSE)

Arguments

graph    A matrix of size n x n of class dgCMatrix.
Y        A numeric vector of length n.
funname  Character scalar. Comparison to make (see vertex_covariate_compare).
all      Logical scalar. When FALSE (default) f_i is mean at ego level. Otherwise is fix
          for all i (see details).

Details

For each vertex i the variance is computed as follows

\[(\sum_{j} a_{ij})^{-1} \sum_{j} a_{ij} [f(y_i, y_j) - f_i]^2\]

Where a_{ij} is the ij-th element of graph, f is the function specified in funname, and, if all=FALSE
f_i = \sum_{j} a_{ij} f(y_i, y_j)^2 / \sum_{j} a_{ij}, otherwise f_i = f_j = \frac{1}{n^2} \sum_{i,j} f(y_i, y_j)

This is an auxiliary function for struct_test. The idea is to compute an adjusted measure of
 dissimilarity between vertices, so the closest in terms of f is i to its neighbors, the smaller the relative variance.

Value

A numeric vector of length n.

See Also

struct_test

Other statistics: bass, classify_adopters, cumulative_adopt_count, dgr, exposure, hazard_rate, infection, moran, struct_equiv, threshold, vertex_covariate_dist
exposure

Ego exposure

Description

Calculates exposure to adoption over time via multiple different types of weight matrices. The basic model is exposure to adoption by immediate neighbors (outdegree) at the time period prior to ego’s adoption. This exposure can also be based on (1) incoming ties, (2) structural equivalence, (3) indirect ties, (4) attribute weighted (5) network-metric weighted (e.g., central nodes have more influence), and attribute-weighted (e.g., based on homophily or tie strength).

Usage

exposure(graph, cumadopt, attrs = NULL, alt.graph = NULL, 
     outgoing = getOption("diffnet.outgoing", TRUE), 
     valued = getOption("diffnet.valued", FALSE), normalized = TRUE, 
     groupvar = NULL, self = getOption("diffnet.self"), lags = 0L, ...)

Arguments

graph A dynamic graph (see netdiffuseR-graphs).
cumadopt $n \times T$ matrix. Cumulative adoption matrix obtained from toa_mat
attrs Either a character scalar (if graph is diffnet), or a numeric matrix of size $n \times T$. Weighting for each time, period (see details).
alt.graph Either a graph that should be used instead of graph, or "se" (see details).
outgoing Logical scalar. When TRUE, computed using outgoing ties.
valued Logical scalar. When TRUE weights will be considered. Otherwise non-zero values will be replaced by ones.
normalized Logical scalar. When TRUE, the exposure will be between zero and one (see details).
groupvar Passed to struct_equiv.
self Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
lags Integer scalar. When different from 0, the resulting exposure matrix will be the lagged exposure as specified (see examples).
... Further arguments passed to struct_equiv (only used when alt.graph="se").

Details

Exposure is calculated as follows:

$$E_t = (S_t \times [x_t \circ A_t]) / (S_t \times x_t)$$
Where \( S_t \) is the graph in time \( t \), \( x_t \) is an attribute vector of size \( n \) at time \( t \), \( A_t \) is the \( t \)-th column of the cumulative adopters matrix (a vector of length \( n \) with \( a_{it} = 1 \) if \( i \) has adopted at or prior to \( t \)), \( \odot \) is the kronecker product (element-wise), and \( \times \) is the matrix product.

By default the graph used for this calculation, \( S \), is the social network. Alternatively, in the case of \texttt{diffnet} objects, the user can provide an alternative graph using \texttt{alt.graph}. An example of this would be using \( 1/SE \), the element-wise inverse of the structural equivalence matrix (see example below). Furthermore, if \texttt{alt.graph}="se", the inverse of the structural equivalence is computed via \texttt{struct_equiv} and used instead of the provided graph. Notice that when using a valued graph the option valued should be equal to \texttt{TRUE}, this check is run automatically when running the model using structural equivalence.

If the \texttt{alt.graph} is static, then the function will warn about it and will recycle the graph to compute exposure at each time point.

**An important remark** is that when calculating structural equivalence the function assumes that this is to be done to the entire graph regardless of disconnected communities (as in the case of the medical innovations data set). Hence, structural equivalence for individuals for two different communities may not be zero. If the user wants to calculate structural equivalence separately by community, he should create different \texttt{diffnet} objects and do so (see example below). Alternatively, for the case of \texttt{diffnet} objects, by using the option \texttt{groupvar} (see \texttt{struct_equiv}), the user can provide the function with the name of a grouping variable–which should one in the set of static vertex attributes–so that the algorithm is done by group (or community) instead of in an aggregated way.

If the user does not specifies a particular weighting attribute in \texttt{attrs}, the function sets this as a matrix of ones. Otherwise the function will return an attribute weighted exposure. When graph is of class \texttt{diffnet}, \texttt{attrs} can be a character scalar specifying the name of any of the graph’s attributes, both dynamic and static. See the examples section for a demonstration using degree.

When \texttt{outgoing}=	exttt{FALSE}, \( S \) is replaced by its transposed, so in the case of a social network exposure will be computed based on the incoming ties.

If \texttt{normalize}=	exttt{FALSE} then denominator, \( S_t \times x_t \), is not included. This can be useful when, for example, exposure needs to be computed as a count instead of a proportion. A good example of this can be found at the examples section of the function \texttt{rdiffnet}.

**Value**

A matrix of size \( n \times T \) with exposure for each node.

**Author(s)**

George G. Vega Yon & Thomas W. Valente

**References**


See Also

Other statistics: \texttt{bass, classify_adopters, cumulative_adopt_count, dgr, ego_variance, hazard_rate, infection, moran, struct_equiv, threshold, vertex_covariate_dist}

Examples

# Calculating lagged exposure --------------------------------------------

set.seed(8)
graph <- rdiffnet(20, 4)

expo0 <- exposure(graph)
expo1 <- exposure(graph, lags = 1)

# These should be equivalent
stopifnot(all(expo0[, -4] == expo1[, -1]))  # No stop!

# Calculating the exposure based on Structural Equivalence ---------------
set.seed(113132)
graph <- rdiffnet(100, 10)

SE <- lapply(struct_equiv(graph), "[['", "SE"])
SE <- lapply(SE, function(x) {
    x <- 1/x
    x[!is.finite(x)] <- 0
    x
})

# Recall setting valued equal to TRUE!
expo_se <- exposure(graph, alt.graph=SE , valued=TRUE)

# These three lines are equivalent to:
expo_se2 <- exposure(graph, alt.graph="se", valued=TRUE)
# Notice that we are setting valued=TRUE, but this is not necessary since when
# alt.graph = "se" the function checks this to be set equal to TRUE

# Weighted Exposure using degree ----------------------------------------
eDE <- exposure(graph, attrs=dgr(graph))

# Which is equivalent to
graph["deg"] <- dgr(graph)
eDE2 <- exposure(graph, attr="deg")

# Comparing using incoming edges ----------------------------------------
eIN <- exposure(graph, outgoing=FALSE)

# Structural equivalence for different communities ---------------------
data(medInnovationsDiffNet)

# METHOD 1: Using the c.diffnet method:
# Creating subsets by city
cities <- unique(medInnovationsDiffNet["city"])

diffnet <- medInnovationsDiffNet[medInnovationsDiffNet["city"] == cities[1]]
diffnet["expo_se"] <- exposure(diffnet, alt.graph="se", valued=TRUE)

for (v in cities[-1]) {
  diffnet_v <- medInnovationsDiffNet[medInnovationsDiffNet["city"] == v]
  diffnet_v["expo_se"] <- exposure(diffnet_v, alt.graph="se", valued=TRUE)
  diffnet <- c(diffnet, diffnet_v)
}

# We can set the original order (just in case) of the data
diffnet <- diffnet[medInnovationsDiffNet$meta$ids]

diffnet

# Checking everything is equal
test <- summary(medInnovationsDiffNet, no.print=TRUE) ==
  summary(diffnet, no.print=TRUE)
stopifnot(all(test[!is.na(test)]))

# METHOD 2: Using the 'groupvar' argument
# Further, we can compare this with using the groupvar
diffnet["expo_se2"] <- exposure(diffnet, alt.graph="se",
  groupvar="city", valued=TRUE)

# These should be equivalent
test <- diffnet["expo_se", as.df=TRUE] == diffnet["expo_se2", as.df=TRUE]
stopifnot(all(test[!is.na(test)]))

# METHOD 3: Computing exposure, rbind and then adding it to the diffnet object
expo_se3 <- NULL
for (v in unique(cities))
  expo_se3 <- rbind(
    expo_se3,
    exposure(
      diffnet[diffnet["city"] == v],
      alt.graph = "se", valued=TRUE
    )
  )

# Just to make sure, we sort the rows
expo_se3 <- expo_se3[diffnet$meta$ids,]

diffnet["expo_se3"] <- expo_se3

test <- diffnet["expo_se", as.df=TRUE] == diffnet["expo_se3", as.df=TRUE]
stopifnot(all(test[!is.na(test)]))

# METHOD 4: Using the groupvar in struct_equiv
se <- struct_equiv(diffnet, groupvar="city")
se <- lapply(se, c("", "SE"))
fakeDynEdgelist

Description
A data frame used for examples in reading edgelist format networks. This edgelist can be merged with the dataset fakesurveyDyn.

Format
A data frame with 22 rows and 4 variables
- **ego** Nominating individual
- **alter** Nominated individual
- **value** Strength of the tie
- **time** Integer with the time of the spell

Author(s)
George G. Vega Yon

Source
Generated for the package

See Also
Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, kfamily, medInnovationsDiffNet, medInnovations
fakeEdgelist  

*Fake static edgelist*

**Description**

A data frame used for examples in reading edgelist format networks. This edgelist can be merged with the dataset `fakesurvey`.

**Format**

A data frame with 11 rows and 3 variables

- **ego**: Nominating individual
- **alter**: Nominated individual
- **value**: Strength of the tie

**Author(s)**

George G. Vega Yon

**Source**

Generated for the package

**See Also**

Other diffusion datasets: `brfarmersDiffNet`, `brfarmers`, `diffusion-data`, `fakeDynEdgelist`, `fakesurveyDyn`, `fakesurvey`, `kfamilyDiffNet`, `kfamily`, `medInnovationsDiffNet`, `medInnovations`

fakesurvey  

*Fake survey data*

**Description**

This data frame is used to illustrate some of the functions of the package, in particular, the `survey_to_diffnet` function. This dataset can be merged with the `fakeEdgelist`.

**Format**

A data frame with 9 rows and 9 variables

- **id**: Unique id at group level
- **toa**: Time of adoption
- **group**: Group id
- **net1**: Network nomination 1
fakesurveyDyn

- net2  Network nomination 2
- net3  Network nomination 3
- age   Age of the respondent
- gender  Gender of the respondent
- note  Description of the respondent

Author(s)
George G. Vega Yon

Source
Generated for the package.

See Also
Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, kfamilyDiffNet, kfamily, medInnovationsDiffNet, medInnovations

| fakesurveyDyn | Fake longitudinal survey data |

Description
This data frame is used to illustrate some of the functions of the package, in particular, the `survey_to_diffnet` function. This dataset can be merged with the `fakeDynEdgelist`.

Format
A data frame with 18 rows and 10 variables

- id  Unique id at group level
- toa  Time of adoption
- group  Group id
- net1  Network nomination 1
- net2  Network nomination 2
- net3  Network nomination 3
- age  Age of the respondent
- gender  Gender of the respondent
- note  Description of the respondent
- time  Timing of the wave

Author(s)
George G. Vega Yon
grid_distribution

Description

Distribution of pairs over a grid of fix size.

Usage

grid_distribution(x, y, nlevels = 100L)

Arguments

- x: Numeric vector of size n
- y: Numeric vector of size n
- nlevels: Integer scalar. Number of bins to return

Details

This function ment for internal use only.

Value

Returns a list with three elements

- x: Numeric vector of size nlevels with the class marks for x
- y: Numeric vector of size nlevels with the class marks for y
- z: Numeric matrix of size nlevels by nlevels with the distribution of the elements in terms of frequency

Examples

# Generating random vectors of size 100
x <- rnorm(100)
y <- rnorm(100)

# Calculating distribution
grid_distribution(x, y, 20)
**hazard_rate**

See Also

- Used by `plot_infectsuscep`
- Other visualizations: `dgr`, `diffusionMap`, `drawColorKey`, `hazard_rate`, `plot_adopters`, `plot_diffnet2`, `plot_diffnet`, `plot_infectsuscep`, `plot_threshold`, `rescale_vertex_igraph`

---

**hazard_rate**  
*Network Hazard Rate*

**Description**

The hazard rate is the instantaneous probability of adoption at each time representing the likelihood members will adopt at that time (Allison 1984). The shape of the hazard rate indicates the pattern of new adopters over time. Rapid diffusion with convex cumulative adoption curves will have hazard functions that peak early and decay over time whereas slow concave cumulative adoption curves will have hazard functions that are low early and rise over time. Smooth hazard curves indicate constant adoption whereas those that oscillate indicate variability in adoption behavior over time.

**Usage**

```r
hazard_rate(obj, no.plot = FALSE, include.grid = TRUE, ...)
```

```r
plot_hazard(x, ...)
```

```r
## S3 method for class 'diffnet_hr'
plot(x, y = NULL, main = "Hazard Rate",
     xlab = "Time", ylab = "Hazard Rate", type = "b",
     include.grid = TRUE, bg = "lightblue", pch = 21, add = FALSE,
     ylim = c(0, 1), ...)
```

**Arguments**

- `obj`  
  A $n \times T$ matrix (Cumulative adoption matrix obtained from `toa_mat`) or a `diffnet` object.
- `no.plot`  
  Logical scalar. When TRUE, suppress plotting (only returns hazard rates).
- `include.grid`  
  Logical scalar. When TRUE includes a grid on the plot.
- `...`  
  Further arguments to be passed to the method.
- `x`  
  An object of class `diffnet_hr`.
- `y`  
  Ignored.
- `main`  
  Character scalar. Title of the plot
- `xlab`  
  Character scalar. x-axis label.
- `ylab`  
  Character scalar. y-axis label.
- `type`  
  Character scalar. See `par`.
- `bg`  
  Character scalar. Color of the points.
\( pch \) Integer scalar. See `par`.

\( add \) Logical scalar. When TRUE it adds the hazard rate to the current plot.

\( ylim \) Numeric vector. See `plot`.

**Details**

This function computes hazard rate, plots it and returns the hazard rate vector invisible (so is not printed on the console). For \( t > 1 \), hazard rate is calculated as

\[
\frac{q_t - q_{t-1}}{n - q_{t-1}}
\]

where \( q_t \) is the number of adopters in time \( t \), and \( n \) is the number of vertices in the graph.

In survival analysis, hazard rate is defined formally as

\[
\lambda(t) = \lim_{h \to +0} \frac{F(t + h) - F(t)}{h} \frac{1}{1 - F(t)}
\]

Then, by approximating \( h = 1 \), we can rewrite the equation as

\[
\lambda(t) = \frac{F(t + 1) - F(t)}{1 - F(t)}
\]

Furthermore, we can estimate \( F(t) \), the probability of not having adopted the innovation in time \( t \), as the proportion of adopters in that time, this is \( F(t) \sim q_t/n \), so now we have

\[
\lambda(t) = \frac{q_t/n}{1 - q_t/n} \frac{q_t + 1/n - q_t/n}{1 - q_t/n} = \frac{q_t + 1 - q_t/n}{n - q_t}
\]

As showed above.

The `plot_hazard` function is an alias for the `plot.diffnet_hr` method.

**Value**

A row vector of size \( T \) with hazard rates for \( t > 1 \) of class `diffnet_hr`. The class of the object is only used by the S3 plot method.

**Author(s)**

George G. Vega Yon & Thomas W. Valente

**References**


Coercion between graph classes

Description

Coercion between graph classes

Usage

diffnet_to_igraph(graph, slices = 1:nsllices(graph))

igraph_to_diffnet(graph = NULL, graph.list = NULL, toavar, t0 = NULL, t1 = NULL, ...

Arguments

- `graph`: Either a `diffnet` or `igraph` graph object.
- `slices`: An integer vector indicating the slices to subset.
- `graph.list`: A list of `igraph` objects.
- `toavar`: Character scalar. Name of the attribute that holds the times of adoption.
- `t0`: Integer scalar. Passed to `new_diffnet`.
- `t1`: Integer scalar. Passed to `new_diffnet`.
- `...`: Further arguments passed to `as_diffnet`.

Value

Either a list of length(slices) `igraph` (`diffnet_to_igraph`), or a `diffnet` object (`igraph_to_diffnet`) objects.

See Also

Other statistics: `bass`, `classify_adaptors`, `cumulative_adopt_count`, `dgr`, `ego_variance`, `exposure`, `infection`, `moran`, `struct_equiv`, `threshold`, `vertex_covariate_dist`

Other visualizations: `dgr`, `diffusionMap`, `drawColorKey`, `grid_distribution`, `plot_adopters`, `plot_diffnet2`, `plot_diffnet`, `plot_infectionsuscep`, `plot_threshold`, `rescale_vertex_igraph`

Examples

`# Creating a random vector of times of adoption`

`# Computing cumulative adoption matrix`
`cumadopt <- toa_mat(toa)$cumadopt`

`# Visualizing the hazard rate`
`hazard_rate(cumadopt)`
See Also

Other Foreign: network, read_pajek, read_ucinet_head

Examples

# Reading the medical innovation data into igraph
x <- diffnet_to_igraph(medInnovationsDiffNet)

# Fetching the times of adoption
igraph::vertex_attr(x[[1]], "toa")


infection

Susceptibility and Infection

Description

Calculates infectiousness and susceptibility for each node in the graph

Usage

infection(graph, toa, t0 = NULL, normalize = TRUE, K = 1L, r = 0.5, expdiscount = FALSE, valued = getOption("diffnet.valued", FALSE), outgoing = getOption("diffnet.outgoing", TRUE))

susceptibility(graph, toa, t0 = NULL, normalize = TRUE, K = 1L, r = 0.5, expdiscount = FALSE, valued = getOption("diffnet.valued", FALSE), outgoing = getOption("diffnet.outgoing", TRUE))

Arguments

graph A dynamic graph (see netdiffuserR-graphs).
toa Integer vector of length n with the times of adoption.
t0 Integer scalar. See toa_mat.
normalize Logical. Whether or not to normalize the outcome
K Integer scalar. Number of time periods to consider
r Numeric scalar. Discount rate used when expdiscount=TRUE
expdiscount Logical scalar. When TRUE, exponential discount rate is used (see details).
valued Logical scalar. When TRUE weights will be considered. Otherwise non-zero values will be replaced by ones.
outgoing Logical scalar. When TRUE, computed using outgoing ties.
Details

Normalization, normalize=TRUE, is applied by dividing the resulting number from the infectiousness/susceptibility stat by the number of individuals who adopted the innovation at time $t$.

Given that node $i$ adopted the innovation in time $t$, its Susceptibility is calculated as follows

$$S_i = \frac{\sum_{k=1}^{K} \sum_{j=1}^{n} x_{ij}(t-k+1)z_{j}(t-k) \times \frac{1}{w_k}}{\sum_{k=1}^{K} \sum_{j=1}^{n} x_{ij}(t-k+1)z_{j}(1\leq t \leq t-k) \times \frac{1}{w_k}} \quad \text{for } i, j = 1, \ldots, n \quad i \neq j$$

where $x_{ij}(t-k+1)$ is 1 whenever there’s a link from $i$ to $j$ at time $t - k + 1$, $z_{j}(t-k)$ is 1 whenever individual $j$ adopted the innovation at time $t - k$, $z_{j}(1\leq t \leq t-k)$ is 1 whenever $j$ had adopted the innovation up to $t - k$, and $w_k$ is the discount rate used (see below).

Similarly, infectiousness is calculated as follows

$$I_i = \frac{\sum_{k=1}^{K} \sum_{j=1}^{n} x_{ji}(t+k-1)z_{j}(t+k) \times \frac{1}{w_k}}{\sum_{k=1}^{K} \sum_{j=1}^{n} x_{ji}(t+k-1)z_{j}(t+k\leq T) \times \frac{1}{w_k}} \quad \text{for } i, j = 1, \ldots, n \quad i \neq j$$

It is worth noticing that, as we can see in the formulas, while susceptibility is from alter to ego, infection is from ego to alter.

When outgoing=FALSE the algorithms are based on incoming edges, this is the adjacency matrices are transposed swapping the indexes $(i, j)$ by $(j, i)$. This can be useful for some users.

Finally, by default both are normalized by the number of individuals who adopted the innovation in time $t - k$. Thus, the resulting formulas, when normalize=TRUE, can be rewritten as

$$S'_i = \frac{S_i}{\sum_{k=1}^{K} \sum_{j=1}^{n} z_{j}(t-k) \times \frac{1}{w_k}} \quad I'_i = \frac{I_i}{\sum_{k=1}^{K} \sum_{j=1}^{n} z_{j}(t-k) \times \frac{1}{w_k}}$$

For more details on these measurements, please refer to the vignette titled *Time Discounted Infection and Susceptibility*.

Value

A numeric column vector (matrix) of size $n$ with either infection/susceptibility rates.

Discount rate

Discount rate, $w_k$ in the formulas above, can be either exponential or linear. When expdiscount=TRUE, $w_k = (1 + r)^{k-1}$, otherwise it will be $w_k = k$.

Note that when $K = 1$, the above formulas are equal to the ones presented in Valente et al. (2015).

Author(s)

George G. Vega Yon
References

Thomas W. Valente, Stephanie R. Dyal, Kar-Hai Chu, Heather Wipfli, Kayo Fujimoto Diffusion of innovations theory applied to global tobacco control treaty ratification, Social Science & Medicine, Volume 145, November 2015, Pages 89-97, ISSN 0277-9536 http://dx.doi.org/10.1016/j.socscimed.2015.10.001


See Also

The user can visualize the distribution of both statistics by using the function plot_infectsuscep

Other statistics: bass, classify_adaptors, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, moran, struct_equiv, threshold, vertex_covariate_dist

Examples

# Creating a random dynamic graph
set.seed(943)
graph <- rgraph_er(n=100, t=10)
toa <- sample.int(10, 100, TRUE)

# Computing infection and susceptibility (K=1)
infection(graph, toa)
susceptibility(graph, toa)

# Now with K=4
infection(graph, toa, K=4)
susceptibility(graph, toa, K=4)

isolated Find and remove isolated vertices

Description

Find and remove unconnected vertices from the graph.

Usage

isolated(graph, undirected = getOption("diffnet.undirected", FALSE),
  self = getOption("diffnet.self", FALSE))

drop_isolated(graph, undirected = getOption("diffnet.undirected", FALSE),
  self = getOption("diffnet.self", FALSE))
isolated

Arguments

- **graph**: Any class of accepted graph format (see `netdiffuserR-graphs`).
- **undirected**: Logical scalar. When TRUE only the lower triangle of the adjacency matrix will be considered (faster).
- **self**: Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).

Value

When `graph` is an adjacency matrix:

- **isolated**: an matrix of size $n \times 1$ with 1's where a node is isolated
- **drop_isolated**: a modified graph excluding isolated vertices.

Otherwise, when `graph` is a list

- **isolated**: an matrix of size $n \times T$ with 1's where a node is isolated
- **drop_isolated**: a modified graph excluding isolated vertices.

Author(s)

George G. Vega Yon

See Also

Other data management functions: `diffnet-class`, `edgelist_to_adjmat`, `egonet_attrs`, `survey_to_diffnet`

Examples

```r
# Generating random graph
set.seed(123)
adjmat <- rgraph_er()

# Making nodes 1 and 4 isolated
adjmat[c(1,4),] <- 0
adjmat[,c(1,4)] <- 0
adjmat

# Finding isolated nodes
iso <- isolated(adjmat)
iso

# Removing isolated nodes
drop_isolated(adjmat)

# Now with a dynamic graph
graph <- rgraph_er(n=10, t=3)

# Making 1 and 5 isolated
```
Description

From Valente (1995) “Scholars at Seoul National University’s School of Public Health (Park, Chung, Han & Lee, 1974) collected data on the adoption of family planning methods among all married women of child-bearing age 25 in Korea villages in 1973 (N = 1,047).”

Format

A data frame with 1,047 rows and 432 columns:

- **village**: Village of residence
- **id**: Respondent ID number
- **recon1**: Card number NA
- **studno1**: Study number NA
- **area1**: Village of residence
- **id1**: Respondent ID number
- **nma1**: Number males age 0
- **nma2**: Number males age 0-4
- **nma3**: Number males age 5-9
- **nma4**: Number males age 10-14
- **nma5**: Number males age 15-19
- **nma6**: Number males age 20-24
- **nma7**: Number males age 25-29
- **nma8**: Number males age 30-34
- **nma9**: Number males age 35-39
- **nma10**: Number males age 40-44
- **nma11**: Number males age 45-49
- **nma12**: Number males age 50-54
- **nma13**: Number males age 55-59
- **nma14**: Number males age 60-64
- **nma15**: Number males age 65-69
nma.ge16  Number males age 70-74
nma.ge17  Number males age 75-79
nma.ge18  Number males age 80+
nfage1   Number females age 0
nfa.ge2   Number females age 0-4
nfa.ge3   Number females age 5-9
nfa.ge4   Number females age 10-14
nfa.ge5   Number females age 15-19
nfa.ge6   Number females age 20-24
nfa.ge7   Number females age 25-29
nfa.ge8   Number females age 30-34
nfa.ge9   Number females age 35-39
nfa.ge10  Number females age 40-44
nfa.ge11  Number females age 45-49
nfa.ge12  Number females age 50-54
nfa.ge13  Number females age 55-59
nfa.ge14  Number females age 60-64
nfa.ge15  Number females age 65-69
nfa.ge16  Number females age 70-74
nfa.ge17  Number females age 75-79
nfa.ge18  Number females age 80+
pregs    total pregnancies
pregs1   number normal deliveries
pregs2   number of induced abortions
pregs3   number of spontaneous abortions
pregs4   number of still births
pregs5   number of deaths after live birth
pregs6   currently pregnant
sons     number of sons
daughts  number of daughters
planning Ever heard of FP or birth control
loop1    Awareness of Loop
loop2    Detailed knowledge of Loop
loop3    Attitudes toward Loop
loop4    Knowledge of Loop used by neighbors
loop5    Knowledge of place of service for Loop
pill1    Awareness of Pill
Detailed knowledge of Pill
Attitudes toward Pill
Knowledge of Pill used by neighbors
Knowledge of place of service for Pill
Awareness of Vasectomy
Detailed knowledge of Vasectomy
Attitudes toward Vasectomy
Knowledge of Vasectomy used by neighbors
Knowledge of place of service for Vasectomy
Awareness of Condoms
Detailed knowledge Condoms
Attitudes toward Condoms
Knowledge of Condoms used by neighbors
Knowledge of place of service for Condoms
Awareness of Rhythm
Detailed knowledge Rhythm
Attitudes toward Rhythm
Knowledge of Rhythm used by neighbors
Awareness of Basic Body Temperature
Detailed knowledge Basic Body Temperature
Attitudes toward BBT
Knowledge of BBT used by neighbors
Awareness of Diaphragm
Detailed knowledge Diaphragm
Attitudes toward Diaphragm
Knowledge of Diaphragm used by neighbors
Awareness of Withdrawal
Detailed knowledge Withdrawal
Attitudes toward Withdrawal
Knowledge of Withdrawal used by neighbors
Awareness of Tubal Ligation
Detailed knowledge TL
Attitudes toward TL
tuba4  Knowledge of TL used by neighbors
fp1   Experience with an FP practice
fp2   Reasons for not practicing
fp3   What would you do if problem was solved
fp4   Any other reason for not practicing
fp5   Reasons for practicing
fp6   time between decision and adoption
fp7   reasons for time lag
fp8   Ever discontinued practicing
fp9   Reasons for discontinuing
fp10  Attitude toward FP
child1 Ideal number of sons
child2 Ideal number of daughters
child3 Ideal number of children regardless of sex
child4 what do if kept having girls
comop1 Spousal communication on # of children
comop2 Spousal communication on FP
comop3 Consensus on opinion between couple
comop4 What was the difference
comop5 Opinion on who should practice
comop6 Different opinions on who should practice
comop7 Who should make final decision
comop8 Residence in old age
net11 Neighbors talk to about FP- 1
net12 Neighbors talk to about FP- 2
net13 Neighbors talk to about FP- 3
net14 Neighbors talk to about FP- 4
net15 Neighbors talk to about FP- 5
famawe1 Family members of FP Practice
famawe2 Parents awareness of FP Practice
famawe3 How did parents-in-law become aware
famawe4 How did parents become aware
famawe5 How did husband become aware
advic1 Advice given to neighbors where to go
advic2 Advice given on method
advic3 Ever met persons who give advice on FP
advic4 Credibility of person advising on FP
advic5  Counter advice given to others
rumor1  Rumors on Loop
rumor2  Rumors on Pill
rumor3  Rumors on Vasectomy
rumor4  Rumors on Condom
rumor5  Rumors on Tuballigation
media1  Possession of Radio
media2  Possession of TV
media3  Subscription to Newspaper
media4  Subscription to Happy Home
media5  Subscription to other magazine
media6  Radio exposure to FP
media7  TV exposure to FP
media8  Daily paper exposure to FP
media9  Happy Home exposure to FP
media10  Magazine exposure to FP
media11  Movie or slide exposure to FP
media12  Poster exposure to FP
media13  Pamphlet exposure to FP
media14  FP Meeting exposure to FP
recno3  Record number NA
studno3  Study number NA
area3  village
id3  id
media15  Public lecture exposure to FP
media16  Mobile van exposure to FP
media17  Neighbors exposure to FP
media18  Workers home visiting exposure to FP
media19  Husband exposure to FP
club1  Awareness of clubs in community
club2  Membership in club
club3  Reasons for not becoming a member
club4  Feeling of necessity of club
club5  Visit of mobile van to area
club6  Service received from van
club7  Decision-making on FP on # children
club8  Decision-making on important goods
club9  Decision-making on childrens discipline
club10 Decision making on purchase wife clothes
net21  Closest neighbor most frequently met
n1adv  Advice received from neighbor 1
n1prac  practice of FP by neighbor 1
net22  Closest neighbor person 2
n2adv  Advice received from neighbor 2
n2prac Practice of FP by neighbor 2
net23  Closest neighbor person 3
n3adv  Advice received from neighbor 3
n3prac Practice of FP by neighbor 3
net24  Closest neighbor 4
n4adv  Advice received from neighbor 4
n4prac Practice of FP by neighbor 4
net25  Closest neighbor 5
n5adv  Advice received from neighbor 5
n5prac Practice of FP by neighbor 5
stand Standard living of above neighbors
educ  Education level of named neighbors
net31  Advice on FP sought from 1
net32  Advice on FP sought from 2
net33  Advice on FP sought from 3
net34  Advice on FP sought from 4
net35  Advice on FP sought from 5
net41  Information provided on FP by 1
net42  Information provided on FP by 1
net43  Information provided on FP by 1
net44  Information provided on FP by 1
net45  Information provided on FP by 1
net51  Seek advice on induced abortion 1
net52  Seek advice on induced abortion 2
net53  Seek advice on induced abortion 3
net54  Seek advice on induced abortion 4
net55  Seek advice on induced abortion 5
age  Age of respondent
agemar Age at first marriage
recno4  Rec no NA
studno4  Study no NA
area4  village
id4  id

net61  Advice on health sought from 1
net62  Advice on health sought from 2
net63  Advice on health sought from 3
net64  Advice on health sought from 4
net65  Advice on health sought from 5
net71  Advice on purchase of goods 1
net72  Advice on purchase of goods 2
net73  Advice on purchase of goods 3
net74  Advice on purchase of goods 4
net75  Advice on purchase of goods 5
net81  Advice on childrens education 1
net82  Advice on childrens education 2
net83  Advice on childrens education 3
net84  Advice on childrens education 4
net85  Advice on childrens education 5
rfampl1  Advice on FP sought by 1
rfampl2  Advice on FP sought by 2
rfampl3  Advice on FP sought by 3
rfampl4  Advice on FP sought by 4
rfampl5  Advice on FP sought by 5
rfampl  Leadership score - indegree FP
rabort1  Advice on abortion sought by 1
rabort2  Advice on abortion sought by 2
rabort3  Advice on abortion sought by 3
rabort4  Advice on abortion sought by 4
rabort5  Advice on abortion sought by 5
rabortl  Leadership score - indegree abortion
rhealth1  Advice on health sought by 1
rhealth2  Advice on health sought by
rhealth3  Advice on health sought by
rhealth4  Advice on health sought by
rhealth5  Advice on health sought by
rhealthl  Leadership score - indegree health
recno5  rec no NA


studno5 study no NA
area5 village
id5 id
rgoods1 Advice on purchases sought by 1
rgoods2 Advice on purchases sought by 2
rgoods3 Advice on purchases sought by 3
rgoods4 Advice on purchases sought by 4
rgoods5 Advice on purchases sought by 5
rgoods1 Leadership score - indegree purchases
reduc1 Advice on education sought by 1
reduc2 Advice on education sought by 2
reduc3 Advice on education sought by 3
reduc4 Advice on education sought by 4
reduc5 Advice on education sought by 5
reduc1 Leadership score - indegree education
hub1 Husbands friend 1
hub2 Husbands friend 2
hub3 Husbands friend 3
hub4 Husbands friend 4
hub5 Husbands friend 5
hubed Husbands education
wifeed Wifes education
wiferel Wifes religion
hubocc Husbands occupation
wifoocc Wifes occupation
know1 Can you insert a loop yourself
know2 Can you remove it alone
know3 Can a man use a loop
know4 How long can a loop be used
know5 Which doctor
know6 Doctor or nurse
know7 Oral pill method
know8 Can men take pills
know9 Long term use
know10 Time required for vasectomy
know11 Does vasectomy = castration
know12 Can any doctor do vasectomies
pref1 Who prefer use: Husband or wife
pref2 Reasons for preferring FP practice by wife
pref3 Reasons for preferring FP practice by husband
ageend Ideal age to end childbearing
cfp Current status of FP
cfatt1 Husbands attitude
cfatt2 In-laws attitude
cfatt3 Own parents attitude
cbyr Start of period from year
cbmnth Start of period from month
ceyr End of period year
cemnth End of period month
clngth Length of period
cawe1 FP contact
cawe2 Awareness of contraceptive method at the time
cawe3 Awareness of service site
cawe4 Credibility
recno6 rec no NA
studno6 study no NA
area6 village
id6 id
fpt1 FP Status time 1
fatt1t1 Husbands attitude T1
fatt2t1 In-laws attitude T1
fatt3t1 Own parents attitude T1
byrt1 Start of Time 1 from year
lngtht1 Length of Time 1
awe1t1 FP Contact Time 1
awe2t1 Methods known at Time 1
awe3t1 Knowledge of service sites Time 1
awe4t1 Credibility of service site Time 1
fpt2 FP Status time 2
fatt1t2 Husbands attitude T2
fatt2t2 In-laws attitude T2
fatt3t2 Own parents attitude T2
byrt2 Start of Time 2 from year
lngtht2 Length of Time 2
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awe1t2</td>
<td>FP Contact Time 2</td>
</tr>
<tr>
<td>awe2t2</td>
<td>Methods known at Time 2</td>
</tr>
<tr>
<td>awe3t2</td>
<td>Knowledge of service sites Time 2</td>
</tr>
<tr>
<td>awe4t2</td>
<td>Credibility of service site Time 2</td>
</tr>
<tr>
<td>fpt3</td>
<td>FP Status time 3</td>
</tr>
<tr>
<td>fatt1t3</td>
<td>Husbands attitude T3</td>
</tr>
<tr>
<td>fatt2t3</td>
<td>In-laws attitude T3</td>
</tr>
<tr>
<td>fatt3t3</td>
<td>Own parents attitude T3</td>
</tr>
<tr>
<td>byrt3</td>
<td>Start of Time 3 from year</td>
</tr>
<tr>
<td>lnghtt3</td>
<td>Length of Time 3</td>
</tr>
<tr>
<td>awe1t3</td>
<td>FP Contact Time 3</td>
</tr>
<tr>
<td>awe2t3</td>
<td>Methods known at Time 3</td>
</tr>
<tr>
<td>awe3t3</td>
<td>Knowledge of service sites Time 3</td>
</tr>
<tr>
<td>awe4t3</td>
<td>Credibility of service site Time 3</td>
</tr>
<tr>
<td>fpt4</td>
<td>FP Status time 4</td>
</tr>
<tr>
<td>fatt1t4</td>
<td>Husbands attitude T4</td>
</tr>
<tr>
<td>fatt2t4</td>
<td>In-laws attitude T4</td>
</tr>
<tr>
<td>fatt3t4</td>
<td>Own parents attitude T4</td>
</tr>
<tr>
<td>byrt4</td>
<td>Start of Time 4 from year</td>
</tr>
<tr>
<td>lnghtt4</td>
<td>Length of Time 4</td>
</tr>
<tr>
<td>awe1t4</td>
<td>FP Contact Time 4</td>
</tr>
<tr>
<td>awe2t4</td>
<td>Methods known at Time 4</td>
</tr>
<tr>
<td>awe3t4</td>
<td>Knowledge of service sites Time 4</td>
</tr>
<tr>
<td>awe4t4</td>
<td>Credibility of service site Time 4</td>
</tr>
<tr>
<td>fpt5</td>
<td>FP Status time 5</td>
</tr>
<tr>
<td>fatt1t5</td>
<td>Husbands attitude T5</td>
</tr>
<tr>
<td>fatt2t5</td>
<td>In-laws attitude T5</td>
</tr>
<tr>
<td>fatt3t5</td>
<td>Own parents attitude T5</td>
</tr>
<tr>
<td>byrt5</td>
<td>Start of Time 5 from year</td>
</tr>
<tr>
<td>lnghtt5</td>
<td>Length of Time 5</td>
</tr>
<tr>
<td>awe1t5</td>
<td>FP Contact Time 5</td>
</tr>
<tr>
<td>awe2t5</td>
<td>Methods known at Time 5</td>
</tr>
<tr>
<td>awe3t5</td>
<td>Knowledge of service sites Time 5</td>
</tr>
<tr>
<td>awe4t5</td>
<td>Credibility of service site Time 5</td>
</tr>
<tr>
<td>fpt6</td>
<td>FP Status time 6</td>
</tr>
<tr>
<td>fatt1t6</td>
<td>Husbands attitude T6</td>
</tr>
<tr>
<td>fatt2t6</td>
<td>In-laws attitude T6</td>
</tr>
</tbody>
</table>
fatt3t6  Own parents attitude T6
byrt6  Start of Time 6 from year
lnght6  Length of Time 6
awe1t6  FP Contact Time 6
awe2t6  Methods known at Time 6
awe3t6  Knowledge of service sites Time 6
awe4t6  Credibility of service site Time 6
recono7  rec no NA
studno7  study no NA
area7  village
id7  id
fpt7  FP Status time 7
fatt1t7  Husbands attitude T7
fatt2t7  In-laws attitude T7
fatt3t7  Own parents attitude T7
byrt7  Start of Time 7 from year
lnght7  Length of Time 7
awe1t7  FP Contact Time 7
awe2t7  Methods known at Time 7
awe3t7  Knowledge of service sites Time 7
awe4t7  Credibility of service site Time 7
fpt8  FP Status time 8
fatt1t8  Husbands attitude T8
fatt2t8  In-laws attitude T8
fatt3t8  Own parents attitude T8
byrt8  Start of Time 8 from year
lnght8  Length of Time 8
awe1t8  FP Contact Time 8
awe2t8  Methods known at Time 8
awe3t8  Knowledge of service sites Time 8
awe4t8  Credibility of service site Time 8
fpt9  FP Status time 9
fatt1t9  Husbands attitude T9
fatt2t9  In-laws attitude T9
fatt3t9  Own parents attitude T9
byrt9  Start of Time 9 from year
lnght9  Length of Time 9
awe1t9  FP Contact Time 9
awe2t9  Methods known at Time 9
awe3t9  Knowledge of service sites Time 9
awe4t9  Credibility of service site Time 9
fpt10  FP Status time 10
fatt1t10 Husbands attitude T10
fatt2t10 In-laws attitude T10
fatt3t10 Own parents attitude T10
byrt10  Start of Time 10 from year
lngtht10 Length of Time 10
awe1t10 FP Contact Time 10
awe2t10 Methods known at Time 10
awe3t10 Knowledge of service sites Time 10
awe4t10 Credibility of service site Time 10
fpt11  FP Status time 11
fatt1t11 Husbands attitude T11
fatt2t11 In-laws attitude T11
fatt3t11 Own parents attitude T11
byrt11  Start of Time 11 from year
lngtht11 Length of Time 11
awe1t11 FP Contact Time 11
awe2t11 Methods known at Time 11
awe3t11 Knowledge of service sites Time 11
awe4t11 Credibility of service site Time 11
fpt12  FP Status time 12
fatt1t12 Husbands attitude T12
fatt2t12 In-laws attitude T12
fatt3t12 Own parents attitude T12
byrt12  Start of Time 12 from year
lngtht12 Length of Time 12
awe1t12 FP Contact Time 12
awe2t12 Methods known at Time 12
awe3t12 Knowledge of service sites Time 12
awe4t12 Credibility of service site Time 12
ado  adopt times years converted to 1=63
ado1
ado2
ado3
commun Village number
toa  Time of Adoption
study Study (for when multiple diff studies used)
Details

The dataset has 1,047 respondents (women) from 25 communities. Collected during 1973 it spans 11 years of data.

Source

The Korean Family Planning data were stored on a Vax tape that Rogers had given to Marc Granovetter who then gave it to his colleague Roland Soong (see Granovetter & Soong, 1983). Granovetter instructed Song to send the tape to me and I had it loaded on the Vax machine at USC in 1990 and was able to download the data to a PC. The first two datasets were acquired for my dissertation (Valente, 1991) and the third added as I completed my book on Network Models of the Diffusion of Innovations (Valente, 1995; also see Valente, 2005).

References


See Also

Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, medInnovationsDiffNet, medInnovations

---

**Description**

A directed dynamic graph with 1,047 vertices and 11 time periods. The attributes in the graph are static and described in kfamily.

**Format**

A diffnet class object.

**See Also**

Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamily, medInnovationsDiffNet, medInnovations
Non-zero element-wise comparison between two sparse matrices

Description

Taking advantage of matrix sparseness, the function only evaluates `fun` between pairs of elements of `A` and `B` where either `A` or `B` have non-zero values. This can be helpful to implement other binary operators between sparse matrices that may not be implemented in the Matrix package.

Usage

```
matrix_compare(A, B, fun)

compare_matrix(A, B, fun)
```

Arguments

- `A` A matrix of size `n*m` of class `dgCMatrix`.
- `B` A matrix of size `n*m` of class `dgCMatrix`.
- `fun` A function that receives 2 arguments and returns a scalar.

Details

Instead of comparing element by element, the function loops through each matrix non-zero elements to make the comparisons, which in the case of sparse matrices can be more efficient (faster). Algorithmically it can be described as follows:

```r
# Matrix initialization
init ans[n,m];

# Loops through non-zero elements of A
for e_A in E_A:
    ans[e_A] = fun(A[e_A], B[e_A])

# Loops through non-zero elements of B and applying the function
# only if it was not applied while looping in E_A.
for e_B in E_B:
    if (ans[e_B] == Empty)
        ans[e_B] = fun(A[e_B], B[e_B])
```

`compare_matrix` is just an alias for `matrix_compare`.

Value

An object of class `dgCMatrix` of size `n*m`.
See Also

Other dyadic-level comparison functions: `vertex_covariate_compare`, `vertex_covariate_dist`

Examples

```r
# These two should yield the same results

# Creating two random matrices
set.seed(89)
A <- rgraph_ba(t = 9, m = 4)
B <- rgraph_ba(t = 9, m = 4)
A;B

# Comparing
ans0 <- matrix_compare(A, B, function(a, b) (a+b)/2)

ans1 <- matrix(0, ncol=10, nrow=10)
for (i in 1:10)
  for (j in 1:10)
    ans1[i,j] <- mean(c(A[i,j], B[i,j]))

# Are these equal?
all(ans0[] == ans1[]) # Should yield TRUE

# More elaborated example (speed)

## Not run:
set.seed(123123123)
A <- rgraph_ba(t = 5e3, m = 2)
B <- rgraph_ba(t = 5e3, m = 2)

Am <- as.matrix(A)
Bm <- as.matrix(B)

comppfun <- function(a, b)
  ifelse(a > b, a, b)

microbenchmark::microbenchmark(
  diffnet = matrix_compare(A, B, comppfun),
  R = matrix(ifelse(Am > Bm, Am, Bm), ncol=ncol(Am)),
  times = 10
)

## Unit: milliseconds
## expr      min   lq median     uq    max neval
## diffnet 352.7989 355.0193 358.5366 357.7138 364.7604 2493.914 10
## R 1648.9607 1744.6762 2491.2435 1947.4344 2729.1274 6260.011 10

## End(Not run)
```
Description

From Valente (1995) “Coleman, Katz and Menzel from Columbia University’s Bureau of Applied Research studied the adoption of tetracycline by physicians in four Illinois communities in 1954.[...] Tetracycline was a powerful and useful antibiotic just introduced in the mid-1950s”

Format

A data frame with 125 rows and 59 columns:

- **city**: city id
- **id**: sequential respondent id
- **detail**: detail man
- **meet**: meetings, lectures, hospitals
- **coll**: colleagues
- **attend**: attend professional meets
- **proage**: professional age
- **length**: length of reside in community
- **here**: only practice here
- **science**: science versus patients
- **position**: position in home base
- **journ2**: journal subscriptions
- **paadico**: Percent alter adoption date imp
- **ado**: adoption month 1 to 18
- **thresh**: threshold
- **ctl**: corrected tl tl-exp level
- **catbak**: category 1-init 2-marg 3-low tl
- **sourinfo**: source of information
- **origid**: original respondent id
- **adopt**: adoption date 1= 11/53
- **recon**: reconstructed med innov
- **date**: date became aware
- **info**: information source
- **most**: most important info source
- **journ**: journals
- **drug**: drug houses
Details

The collected dataset has 125 respondents (doctors), and spans 17 months of data collected in 1955. Time of adoption of non-adopters has been set to month 18 (see the manual entry titled *Difussion Network Datasets*).
medInnovationsDiffNet

Source

The Medical Innovation data were stored in file cabinets in a basement building at Columbia University. Ron Burt (1987) acquired an NSF grant to develop network diffusion models and retrieve the original surveys and enter them into a database. He distributed copies of the data on diskette and sent one to me, Tom Valente, and I imported onto a PC environment.

References


See Also

Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, kfamily, medInnovationsDiffNet

Description

A directed dynamic graph with 125 vertices and 18 time periods. The attributes in the graph are static and described in medInnovations.

Format

A diffnet class object.

See Also

Other diffusion datasets: brfarmersDiffNet, brfarmers, diffusion-data, fakeDynEdgelist, fakeEdgelist, fakesurveyDyn, fakesurvey, kfamilyDiffNet, kfamily, medInnovations
mentor_matching

Optimal Leader/Mentor Matching

Description
Implementes the algorithm described in Valente and Davis (1999)

Usage

mentor_matching(graph, n, cmode = "indegree",
lead.ties.method = "average", geodist.args = list())

leader_matching(graph, n, cmode = "indegree",
lead.ties.method = "average", geodist.args = list())

## S3 method for class 'diffnet_mentor'
plot(x, y = NULL, vertex.size = "degree",
minmax.relative.size = getOption("diffnet.minmax.relative.size",
c(0.01, 0.04)), lead.cols = grDevices:: topo.colors(attr(x, "nleaders")), vshapes = c(Leader = "square", Follower = "circle"),
add.legend = TRUE, main = "Mentoring Network", ...)

Arguments

graph Any class of accepted graph format (see netdiffuseR-graphs).
n Number of leaders
cmode Passed to dgr.
lead.ties.method Passed to rank
geodist.args Passed to approx_geodesic.
x An object of class diffnet_mentor.
y Ignored.
vertex.size Either a numeric scalar or vector of size n, or any of the following values: "indegree", "degree", or "outdegree" (see details).
minmax.relative.size Passed to rescale_vertex_igraph.
lead.cols Character vector of length attr(x,"nleaders"). Colors to be applied to each group. (see details)
vshapes Character scalar of length 2. Shapes to identify leaders (mentors) and followers respectively.
add.legend Logical scalar. When TRUE generates a legend to distinguish between leaders and followers.
main Character scalar. Passed to title
... Further arguments passed to plot.igraph
Details

The algorithm works as follows:

1. Find the top $n$ individuals ranking them by $dgr(\text{graph}, \text{cmode})$. The rank is computed by the function $\text{rank}$. Denote this set $M$.
2. Compute the geodesic matrix.
3. For each $v$ in $V$ do:
   (a) Find the mentor $m$ in $M$ such that is closest to $v$
   (b) Were there a tie, choose the mentor that minimizes the average path length from $v$'s direct neighbors to $m$.
   (c) If there are no paths to any member of $M$, or all have the same average path length to $v$'s neighbors, then assign one randomly.

Plotting is done via the function $\text{plot.igraph}$.

When $\text{vertex.size}$ is either of "degree", "indegree", or "outdegree", $\text{vertex.size}$ will be replace with $dgr(\cdot, \text{cmode} = )$ so that the vertex size reflects the desired degree.

The argument $\text{minmax.relative.size}$ is passed to $\text{rescale_vertex_igraph}$ which adjusts $\text{vertex.size}$ so that the largest and smallest vertices have a relative size of $\text{minmax.relative.size}[2]$ and $\text{minmax.relative.size}[1]$ respectively with respect to the x-axis.

Value

An object of class $\text{diffnet_mentor}$ and $\text{data.frame}$ with the following columns:

- $\text{name}$ Character. Labels of the vertices
- $\text{degree}$ Numeric. Degree of each vertex in the graph
- $\text{iselader}$ Logical. TRUE when the vertex was picked as a leader.
- $\text{match}$ Character. The corresponding matched leader.

The object also contains the following attributes:

- $\text{nleaders}$ Integer scalar. The resulting number of leaders (could be greater than $n$)
- $\text{graph}$ The original graph used to run the algorithm.

References

Examples

```r
# A simple example -----------------------------------------------
set.seed(1231)
graph <- rgraph_wsn(n=50, k = 4, p = .5)

# Looking for 3 mentors
ans <- mentor_matching(graph, n = 3)

head(ans)
table(ans$match) # We actually got 9 b/c of ties

# Visualizing the mentor network
plot(ans)
```

---

**moran**

*Computes Moran’s I correlation index*

Description

Natively built for computing Moran’s I on `dgCMatrix` objects, this routine allows computing the I on large sparse matrices (graphs). Part of its implementation was based on `ape::Moran.I`, which computes the I for dense matrices.

Usage

```r
moran(x, w, normalize.w = TRUE, alternative = "two.sided")
```

Arguments

- **x** Numeric vector of size \( n \).
- **w** Numeric matrix of size \( n \times n \). Weights. It can be either a object of class `matrix` or `dgCMatrix` from the `Matrix` package.
- **normalize.w** Logical scalar. When TRUE normalizes rowsums to one (or zero).
- **alternative** Character String. Specifies the alternative hypothesis that is tested against the null of no autocorrelation; must be of one “two.sided”, “less”, or “greater”.

Details

In the case that the vector \( x \) is close to constant (degenerate random variable), the statistic becomes irrelevant, and furthermore, the standard error tends to be undefined (NaN).
Value

A list of class diffnet_moran with the following elements:

- observed: Numeric scalar. Observed correlation index.
- expected: Numeric scalar. Expected correlation index equal to $-1/(N-1)$.
- sd: Numeric scalar. Standard error under the null.

Author(s)

George G. Vega Yon

References


See Also

Other statistics: bass, classify_adopters, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, infection, struct_equiv, threshold, vertex_covariate_dist

Other Functions for inference: bootnet, struct_test

Examples

```r
## Not run:

# Generating a small random graph
set.seed(123)
graph <- rgraph_ba(t = 4)
w <- approx_geodesic(graph)
x <- rnorm(5)

# Computing Moran's I
moran(x, w)

# Comparing with the ape's package version
ape::Moran.I(x, as.matrix(w))

## End(Not run)
```
Description

Statistical analysis, visualization and simulation of diffusion and contagion processes on networks. The package implements algorithms for calculating stats such as innovation threshold levels, infectiousness (contagion) and susceptibility, and hazard rates as presented in Burt (1987), Valente (1995), and Myers (2000) (among others).

You can access to the project website at https://github.com/USCCANA/netdiffuserR

Details

Analysis of Diffusion and Contagion Processes on Networks

Acknowledgements

netdiffuseR was created with the support of grant R01 CA157577 from the National Cancer Institute/National Institutes of Health.

Workshops and Tutorials

Online you can find several learning resources:

- Sunbelt 2018 https://github.com/USCCANA/netdiffuser-sunbelt2018

Author(s)

George G. Vega Yon & Thomas W. Valente

Description

List of accepted graph formats
Details

The netdiffuseR package can handle different types of graph objects. Two general classes are defined across the package’s functions: static graphs, and dynamic graphs.

- **In the case of static graphs**, these are represented as adjacency matrices of size $n \times n$ and can be either `matrix` (dense matrices) or `dgCMatrix` (sparse matrix from the Matrix package). While most of the package functions are defined for both classes, the default output graph is sparse, i.e. `dgCMatrix`.

- **With respect to dynamic graphs**, these are represented by either a `diffnet` object, an array of size $n \times n \times T$, or a list of size $T$ with sparse matrices (class `dgCMatrix`) of size $n \times n$. Just like the static graph case, while most of the functions accept both graph types, the default output is `dgCMatrix`.

diffnet objects

In the case of `diffnet`-class objects, the following arguments can be omitted when calling functions suitable for graph objects:

- `toa`: Time of Adoption vector
- `adopt`: Adoption Matrix
- `cumadopt`: Cumulative Adoption Matrix
- `undirected`: Whether the graph is directed or not

Objects’ names

When possible, netdiffuseR will try to reuse graphs dimensional names, this is, `rownames`, `colnames`, `dimnames` and `names` (in the case of dynamic graphs as lists). Otherwise, when no names are provided, these will be created from scratch.

Author(s)

George G. Vega Yon

---

Description

netdiffuseR default options

Details

Set of default options used by the package. These can be retrieved via `getOption` using the prefix `diffnet` (see examples)
Value

The full list of options follows:

- undirected: FALSE
- self: FALSE
- multiple: FALSE
- tol: 1e-8 (used for package testing)
- valued: FALSE
- outgoing: TRUE
- keep.isolates: TRUE
- minmax.relative.size: c(0.025, 0.05)

Author(s)

George G. Vega Yon

Examples

```r
getOption("diffnet.undirected")
getOption("diffnet.multiple")
getOption("diffnet.self")
```

Usage

```r
netmatch_prepare(dat, graph, timevar, depvar, covariates, 
treat_thr = rep(1L, length(graph)), adopt_thr = rep(1L, 
length(graph)), expo.pcnt = FALSE, expo.lag = 0L)

netmatch(dat, graph, timevar, depvar, covariates, treat_thr = rep(1L, 
length(graph)), adopt_thr = rep(1L, length(graph)), 
expo.pcnt = FALSE, expo.lag = 0L, ...)
```
Arguments

- **dat**: data.frame with dynamic data. Must be of nrow(dat) == nslices(graph) * nnodes(graph).
- **graph**: List with sparse matrices.
- **timevar**: Character scalar. Name of time variable.
- **depvar**: Character scalar. Name of the dependent variable.
- **covariates**: Character vector. Name(s) of the control variable(s).
- **treat_thr**: Either a numeric scalar or vector of length nslices(graph). Sets the threshold of exposure at which it is considered that an observation is treated.
- **adopt_thr**: Either a numeric scalar or vector of length nslices(graph). Sets the threshold of depvar at which it is considered that an observation has adopted a behavior.
- **expo_pcent**: Logical scalar. When TRUE, exposure is computed non-normalized (so it is a count rather than a percentage).
- **expo_lag**: Integer scalar. Number of lags to consider when computing exposure. expo_lag=1 defines exposure in T considering behavior and network at T-1.
- **...**: Further arguments to be passed to `matchit`.

Details

In Aral et al. (2009), the matching estimator is used as a response to the fact that the observed network is homophilous. Essentially, using exposure as a treatment indicator, which is known to be endogenous, we can apply the same principle of matching estimators in which, after controlling for characteristics (covariates), individuals from the treated group (exposed to some behavior) can be compared to individuals from the control group (not exposed to that behavior), as the only difference between the two is the exposure.

As pointed out in King & Nielsen (2015), it is suggested that, contrary to what Aral et al. (2009), the matching is not performed over propensity score since it is know that the later can increase imbalances in the data and thus obtaining exactly the opposed outcome that matching based estimators pursue.

A couple of good references for matching estimators are Imbens and Wooldridge (2009), and Sekhon (2008).

Value

In the case of `netmatch_prepare`

- **dat**: A data.frame with the original data (covariates), plus the following new variables: treat, adopt, exposure.
- **match_model**: A formula to be passed to `netmatch`

`netmatch` returns the following:

- **fATT**: A numeric vector of length $N_t$ (number of treated used in the matching process). Treatment effects on the treated at the individual level
- **match_obj**: The output from `matchit`
Author(s)
George G. Vega Yon

References

Coercion between `diffnet`, `network` and `networkDynamic`

Usage

diffnet_to_network(graph, slices = 1:nslices(graph), ...)
diffnet_to_networkDynamic(graph, slices = 1:nslices(graph),
  diffnet2net.args = list(), netdyn.args = list())
networkDynamic_to_diffnet(graph, toavar)
network_to_diffnet(graph = NULL, graph.list = NULL, toavar,
  t0 = NULL, t1 = NULL)

Arguments

graph An object of class `diffnet`
slices An integer vector indicating the slices to subset
... Further arguments passed to `networkDynamic`
diffnet2net.args List of arguments passed to `diffnet_to_network`.
netdyn.args List of arguments passed to `networkDynamic`
toavar Character scalar. Name of the vertex attribute that holds the times of adoption.
graph.list A list of `network` objects.
t0 Integer scalar. Passed to `new_diffnet`.
t1 Integer scalar. Passed to `new_diffnet`. 
Details

diffnet_to_networkDynamic calls diffnet_to_network and uses the output to call networkDynamic, passing the resulting list of network objects as network.list (see networkDynamic).

By default, diffnet_to_networkDynamic passes net.obs.period as

```r
net.obs.period = list(
  observations = list(range(graph$meta$pers)),
  mode="discrete",
  time.increment = 1,
  time.unit = "step"
)
```

By default, networkDynamic_to_diffnet uses the first slice as reference for vertex attributes and times of adoption.

By default, network_to_diffnet uses the first element of graph (a list) as reference for vertex attributes and times of adoption.

Value

diffnet_to_network returns a list of length length(slices) in which each element is a network object corresponding a slice of the graph (diffnet object). The attributes list will include toa (time of adoption).

An object of class networkDynamic.

Caveats

Since diffnet does not support edges attributes, these will be lost when converting from network-type objects. The same applies to network attributes.

See Also

Other Foreign: igraph, read_pajek, read_ucinet_head

Examples

# Cohering a diffnet to a list of networks -----------------------------------
set.seed(1)
ans <- diffnet_to_network(rdiffnet(20, 2))
ans

# and back
network_to_diffnet(graph.list = ans, toavar="toa")

# If it was static, we can use -graph- instead
network_to_diffnet(ans[[1]], toavar="toa")

# A random diffusion network ----------------------------------------------
set.seed(87)
dn <- rdiffnet(50, 4)
nvertices <- diffnet_to_networkDynamic(dn)

# and back
networkDynamic_to_diffnet(ans, toavar = "toa")

<table>
<thead>
<tr>
<th>nvertices</th>
<th>Count the number of vertices/edges/slices in a graph</th>
</tr>
</thead>
</table>

**Description**

Count the number of vertices/edges/slices in a graph

**Usage**

nvertices(graph)
nnodes(graph)
nedges(graph)
nlinks(graph)
nslices(graph)

**Arguments**

- **graph**: Any class of accepted graph format (see netdiffuseR-graphs).

**Details**

nnodes and nlinks are just aliases for nvertices and nedges respectively.

**Value**

For nvertices and nslices, an integer scalar equal to the number of vertices and slices in the graph. Otherwise, from nedges, either a list of size t with the counts of edges (non-zero elements in the adjacency matrices) at each time period, or, when graph is static, a single scalar with such number.

**Examples**

# Creating a dynamic graph (we will use this for all the classes) ---------
set.seed(13133)
diffnet <- rdiffnet(100, 4)

# Lets use the first time period as a static graph
graph_mat <- diffnet$graph[[1]]
graph_dgCMatrix <- methods::as(graph_mat, "dgCMatrix")
# Now lets generate the other dynamic graphs
graph_list <- diffnet$graph
graph_array <- as.array(diffnet) # using the as.array method for diffnet objects

# Now we can compare vertices counts
nvertices(diffnet)
nvertices(graph_list)
nvertices(graph_array)
nvertices(graph_mat)
nvertices(graph_dgCMatrix)

# ... and edges count
nedges(diffnet)
nedges(graph_list)
nedges(graph_array)
nedges(graph_mat)
nedges(graph_dgCMatrix)

permute_graph                            Permute the values of a matrix

Description

permute_graph Shuffles the values of a matrix either considering loops and multiple links (which are processed as cell values different than 1/0). rewire_qap generates a new graph graph' that is isomorphic to graph.

Usage

permute_graph(graph, self = FALSE, multiple = FALSE)
rewire_permute(graph, self = FALSE, multiple = FALSE)
rewire_qap(graph)

Arguments

graph Any class of accepted graph format (see netdiffuseR-graphs).
self Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
multiple Logical scalar. When TRUE allows multiple edges.

Value

A permuted version of graph.
Author(s)

George G. Vega Yon

References


See Also

This function can be used as null distribution in struct_test

Other simulation functions: rdiffnet, rewire_graph, rgraph_ba, rgraph_er, rgraph_ws, ring_lattice

Examples

```r
# Simple example
set.seed(1231)
g <- rgraph_ba(t=9)
g

# These preserve the density
permute_graph(g)
permute_graph(g)

# These are isomorphic to g
rewire_qap(g)
rewire_qap(g)
```

Description

S3 plotting method for diffnet objects.

Usage

```r
# S3 method for class 'diffnet'
plot(x, y = NULL, t = 1, vertex.color = c(adopt = "steelblue", noadopt = "white"), vertex.size = "degree",
     main = "Diffusion network in time %d",
     minmax.relative.size = getOption("diffnet.minmax.relative.size", c(0.01, 0.04)), ...)
```
Arguments

x  An object of class diffnet
y  Ignored.
t  Integer scalar indicating the time slice to plot.

vertex.color  Character scalar/vector. Color of the vertices.

vertex.size  Either a numeric scalar or vector of size n, or any of the following values: "in-degree", "degree", or "out-degree" (see details).

main  Character. A title template to be passed to sprintf.

minmax.relative.size
  Passed to rescale_vertex_igraph.

...  Further arguments passed to plot.igraph.

Details

Plotting is done via the function plot.igraph.

When vertex.size is either of "degree", "in-degree", or "out-degree", vertex.size will be replaced with dgr(. , cmode = ) so that the vertex size reflects the desired degree.

The argument minmax.relative.size is passed to rescale_vertex_igraph which adjusts vertex.size so that the largest and smallest vertices have a relative size of minmax.relative.size[2] and minmax.relative.size[1] respectively with respect to the x-axis.

Value

A matrix with the coordinates of the vertices.

Author(s)

George G. Vega Yon

See Also

Other diffnet methods: %*%, as.array.diffnet, c.diffnet, diffnet-arithmetic, diffnet-class, diffnet_index, summary.diffnet

Examples

data(medInnovationsDiffNet)
plot(medInnovationsDiffNet)
plot_adopters  
Visualize adopters and cumulative adopters

Description
Visualize adopters and cumulative adopters

Usage

plot_adopters(obj, freq = FALSE, what = c("adopt", "cumadopt"),
yadd = FALSE, include.legend = TRUE, include.grid = TRUE,
pch = c(21, 24), type = c("b", "b"), ylim = if (!freq) c(0, 1) else
NULL, lty = c(1, 1), col = c("black", "black"), bg = c("tomato",
"gray"), xlab = "Time", ylab = ifelse(freq, "Frequency",
"Proportion"), main = "Adopters and Cumulative Adopters", ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Either a diffnet object or a cumulative adoption matrix.</td>
</tr>
<tr>
<td>freq</td>
<td>Logical scalar. When TRUE frequencies are plotted instead of proportions.</td>
</tr>
<tr>
<td>what</td>
<td>Character vector of length 2. What to plot.</td>
</tr>
<tr>
<td>add</td>
<td>Logical scalar. When TRUE lines and dots are added to the current graph.</td>
</tr>
<tr>
<td>include.legend</td>
<td>Logical scalar. When TRUE a legend of the graph is plotted.</td>
</tr>
<tr>
<td>include.grid</td>
<td>Logical scalar. When TRUE, the grid of the graph is drawn</td>
</tr>
<tr>
<td>pch</td>
<td>Integer vector of length 2. See matplot.</td>
</tr>
<tr>
<td>type</td>
<td>Character vector of length 2. See matplot.</td>
</tr>
<tr>
<td>ylim</td>
<td>Numeric vector of length 2. Sets the plotting limit for the y-axis.</td>
</tr>
<tr>
<td>lty</td>
<td>Numeric vector of length 2. See matplot.</td>
</tr>
<tr>
<td>col</td>
<td>Character vector of length 2. See matplot.</td>
</tr>
<tr>
<td>bg</td>
<td>Character vector of length 2. See matplot.</td>
</tr>
<tr>
<td>xlab</td>
<td>Character scalar. Name of the x-axis.</td>
</tr>
<tr>
<td>ylab</td>
<td>Character scalar. Name of the y-axis.</td>
</tr>
<tr>
<td>main</td>
<td>Character scalar. Title of the plot</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed to matplot.</td>
</tr>
</tbody>
</table>

Value
A matrix as described in cumulative_adopt_count.

Author(s)
George G. Vega Yon
plot_diffnet

See Also

Other visualizations: dgr, diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_diffnet2, plot_diffnet, plot_infectsuscep, plot_threshold, rescale_vertex_igraph

Examples

#Generating a random diffnet ---------------------------------------------
set.seed(821)
diffnet <- rdiffnet(100, 5, seed.graph="small-world", seed.nodes="central")

plot_adopters(diffnet)

# Alternatively, we can use a TOA Matrix
toa <- sample(c(NA, 2010L, 2015L), 20, TRUE)
mat <- toa_mat(toa)
plot_adopters(mat$cumadopt)

plot_diffnet

Plot the diffusion process

Description

Creates a colored network plot showing the structure of the graph through time (one network plot for each time period) and the set of adopter and non-adopters in the network.

Usage

plot_diffnet(...)

## S3 method for class 'diffnet'
plot_diffnet(graph, ...)

## Default S3 method:
plot_diffnet(graph, cumadopt, slices = NULL,
  vertex.color = c("white", "tomato", "steelblue"),
  vertex.shape = c("square", "circle", "circle"),
  vertex.size = "degree", mfrow.par = NULL,
  main = c("Network in period %s", "Diffusion Network"),
  legend.args = list(),
  minmax.relative.size = getOption("diffnet.minmax.relative.size",
  c(0.01, 0.04)), background = NULL, ...)

Arguments

... Further arguments to be passed to plot.igraph.

graph A dynamic graph (see netdiffuseR-graphs).

cumadopt \( n \times T \) matrix.
slices  Integer vector. Indicates what slices to plot. By default all are plotted.
vertex.color  A character vector of size 3 with colors names.
vertex.shape  A character vector of size 3 with shape names.
vertex.size  Either a numeric scalar or vector of size $n$, or any of the following values: "indegree", "degree", or "outdegree" (see details).
mfrow.par  Vector of size 2 with number of rows and columns to be passed to `par`.
main  Character scalar. A title template to be passed to `sprintf`.
legend.args  List of arguments to be passed to `legend`.
minmax.relative.size  Passed to `rescale_vertex_igraph`.
background  Either a function to be called before plotting each slice, a color to specify the backgroupd color, or NULL (in which case nothing is done).

**Details**

Plotting is done via the function `plot.igraph`.

When `vertex.size` is either of "degree", "indegree", or "outdegree", `vertex.size` will be replace with `dgr(., cmode = )` so that the vertex size reflects the desired degree.

The argument `minmax.relative.size` is passed to `rescale_vertex_igraph` which adjusts `vertex.size` so that the largest and smallest vertices have a relative size of `minmax.relative.size[2]` and `minmax.relative.size[1]` respectively with respect to the x-axis.

Plotting is done via the function `plot.igraph`.

In order to center the attention on the diffusion process itself, the positions of each vertex are computed only once by aggregating the networks through time, this is, instead of computing the layout for each time $t$, the function creates a new graph accumulating links through time.

The `mfrow.par` sets how to arrange the plots on the device. If $T = 5$ and `mfrow.par = c(2, 3)`, the first three networks will be in the top of the device and the last two in the bottom.

The argument `vertex.color` contains the colors of non-adopters, new-adopters, and adopters respectively. The new adopters (default color "tomato") have a different color that the adopters when the graph is at their time of adoption, hence, when the graph been plotted is in $t = 2$ and `toa = 2` the vertex will be plotted in red.

`legend.args` has the following default parameter:

```
  x       "bottom"
  legend  c("Non adopters", "New adopters","Adopters")
  pch sapply(vertex.shape, switch, circle = 21, square = 22, 21)
  bty "n"
  horiz TRUE
```

**Value**

Calculated coordinates for the grouped graph (invisible).
Author(s)
George G. Vega Yon

See Also
Other visualizations: dgr, diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet2, plot_infectsuscep, plot_threshold, rescale_vertex_igraph

Examples

# Generating a random graph
set.seed(1234)
n <- 6
nper <- 5
graph <- rgraph_er(n, nper, p=.3, undirected = FALSE)
toa <- sample(2000:(2000+nper-1), n, TRUE)
adopt <- toa_mat(toa)
plot_diffnet(graph, adopt$cumadopt)

plot_diffnet2

Description
Another way of visualizing diffusion

Usage

plot_diffnet2(graph, ...)

## S3 method for class 'diffnet'
plot_diffnet2(graph, toa, slice = nslices(graph), ...)

## Default S3 method:
plot_diffnet2(graph, toa, pers = min(toa, na.rm = TRUE):max(toa, na.rm = TRUE),
              color.ramp = grDevices::colorRamp("viridisLite::magma(R)",
              layout = NULL, key.width = 0.1, key.args = list(),
              main = "Diffusion dynamics", add.map = NULL,
              diffmap.args = list(kde2d.args = list(n = 100)), diffmap.alpha = 0.5,
              include.white = "first", vertex.size = "degree",
              minmax.relative.size =getOption("diffnet.minmax.relative.size",
              c(0.01, 0.04)), no.graph = FALSE, ...)
Arguments

- **graph**: Any class of accepted graph format (see netdiffuserR-graphs).
- **...**: Further arguments passed to `plot.igraph`.
- **toa**: Integer vector of length \( n \) with the times of adoption.
- **slice**: Integer scalar. Number of slice to use as baseline for drawing the graph.
- **pers**: Integer vector of length \( T \) indicating the time periods of the data.
- **color.ramp**: A function as returned by `colorRamp`.
- **layout**: Passed to `plot.igraph`.
- **key.width**: Numeric scalar. Sets the proportion of the plot (x-axis) that the key uses.
- **key.args**: List. Further arguments to be passed to `drawColorKey`.
- **main**: Character scalar. Title of the graph.
- **add.map**: Character scalar. When "first" plots a `diffusionMap` before the graph itself. If "last" then it adds it at the end. When NULL adds nothing.
- **diffmap.args**: List. If `add.map=TRUE`, arguments passed to `diffusionMap`.
- **diffmap.alpha**: Numeric scalar between \([0,1]\). Alpha level for the map.
- **include.white**: Character scalar. Includes white in the color palette used in the map. When `include.white=NULL` then it won’t include it.
- **vertex.size**: Either a numeric scalar or vector of size \( n \), or any of the following values: "in-degree", "degree", or "outdegree" (see details).
- **minmax.relative.size**: Passed to `rescale_vertex_igraph`.
- **no.graph**: Logical scalar. When TRUE the graph is not drawn. This only makes sense when the option `add.map` is active.

Details

Plotting is done via the function `plot.igraph`.

When `vertex.size` is either of "degree", "indegree", or "outdegree", `vertex.size` will be replace with \( \text{dgr(., cmode = )} \) so that the vertex size reflects the desired degree.

The argument `minmax.relative.size` is passed to `rescale_vertex_igraph` which adjusts `vertex.size` so that the largest and smallest vertices have a relative size of `minmax.relative.size[2]` and `minmax.relative.size[1]` respectively with respect to the x-axis.

If `key.width<=0` then no key is created.

By default, the function passes the following values to `plot.igraph`:

- `vertex.label` equals to ""
- `vertex.frame.color` equals to "white"
- `add` equals to TRUE
- `rescale` equals to FALSE
- `vertex.size` equals to `rescale.fun(vertex.size)`
plot_infectsuscep

Value

A list with the following elements

- **layout**: A numeric matrix with vertex coordinates.
- **vertex.color**: A character vector with computed colors for each vertex.
- **vertex.label**: The value passed to plot_diffnet2.
- **vertex.shape**: A character vector with assigned shapes.
- **vertex.size**: A numeric vector with vertices sizes.
- **diffmap**: If add.map=TRUE, the returned values from diffmap.

Author(s)

George G. Vega Yon

See Also

Other visualizations: dgr, diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet, plot_infectsuscep, plot_threshold, rescale_vertex_igraph

plot_infectsuscep  

*Plot distribution of infect/suscep*

Description

After calculating infectiousness and susceptibility of each individual on the network, it creates an nlevels by nlevels matrix indicating the number of individuals that lie within each cell, and draws a heatmap.

Usage

plot_infectsuscep(graph, toa, t0 = NULL, normalize = TRUE, K = 1L, 
                   r = 0.5, expdiscount = FALSE, bins = 20, nlevels = round(bins/2), 
                   h = NULL, logscale = TRUE, 
                   main = "Distribution of Infectiousness and\nSusceptibility", 
                   xlab = "Infectiousness of ego", ylab = "Susceptibility of ego", 
                   sub = ifelse(logscale, "(in log-scale)", NA), 
                   color.palette = function(n) viridisLite::viridis(n), 
                   include.grid = TRUE, exclude.zeros = FALSE, 
                   valued = getOption("diffnet.valued", FALSE), ...)

Arguments

- **graph**: A dynamic graph (see netdiffuseR-graphs).
- **toa**: Integer vector of length n with the times of adoption.
- **t0**: Integer scalar. See toa_mat.
- **normalize**: Logical scalar. Passed to infection/susceptibility.
- **K**: Integer scalar. Passed to infection/susceptibility.
- **r**: Numeric scalar. Passed to infection/susceptibility.
- **expdiscount**: Logical scalar. Passed to infection/susceptibility.
- **bins**: Integer scalar. Size of the grid (n).
- **nlevels**: Integer scalar. Number of levels to plot (see filled.contour).
- **h**: Numeric vector of length 2. Passed to kde2d in the MASS package.
- **logscale**: Logical scalar. When TRUE the axis of the plot will be presented in log-scale.
- **main**: Character scalar. Title of the graph.
- **xlab**: Character scalar. Title of the x-axis.
- **ylab**: Character scalar. Title of the y-axis.
- **sub**: Character scalar. Subtitle of the graph.
- **color.palette**: a color palette function to be used to assign colors in the plot (see filled.contour).
- **include.grid**: Logical scalar. When TRUE, the grid of the graph is drawn.
- **exclude.zeros**: Logical scalar. When TRUE, observations with zero values valued Logical scalar. When FALSE non-zero values in the adjmat are set to one, in infect or suscept are excluded from the graph. This is done explicitly when logscale=TRUE.
- **...**: Additional parameters to be passed to filled.contour.

Details

This plotting function was inspired by Aral, S., & Walker, D. (2012).

By default the function will try to apply a kernel smooth function via kde2d. If not possible (because not enough data points), then the user should try changing the parameter h or set it equal to zero.

toa is passed to infection/susceptibility.

Value

A list with three elements:

- **infect**: A numeric vector of size n with infectiousness levels
- **suscep**: A numeric vector of size n with susceptibility levels
- **coords**: A list containing the class marks and counts used to draw the plot via filled.contour (see grid_distribution)
- **complete**: A logical vector with TRUE when the case was included in the plot. (this is relevant whenever logscale=TRUE)
plot_threshold

Author(s)

George G. Vega Yon

References


See Also

Infectiousness and susceptibility are computed via infection and susceptibility.

Other visualizations: dgr, diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet2, plot_diffnet, plot_threshold, rescale_vertex_igraph

Examples

# Generating a random graph --------------------------------------------
set.seed(1234)
n <- 100
nper <- 20
graph <- rgraph_er(n, nper, p=.2, undirected = FALSE)
toa <- sample(1:(1+nper-1), n, TRUE)

# Visualizing distribution of suscep/infect
gout <- plot_infectsuscep(graph, toa, K=3, logscale = FALSE)

plot_threshold Threshold levels through time

Description

Draws a graph where the coordinates are given by time of adoption, x-axis, and threshold level, y-axis.

Usage

plot_threshold(graph, expo, ...)

## S3 method for class 'diffnet'
plot_threshold(graph, expo, ...)

## S3 method for class 'array'
plot_threshold(graph, expo, ...)

## Default S3 method:
plot_threshold(graph, expo, toa,
include_censored = FALSE, t0 = min(toa, na.rm = TRUE),
plot_threshold

attrs = NULL, undirected = getOption("diffnet.undirected"),
o.contemporary = TRUE,
main = "Time of Adoption by\nNetwork Threshold", xlab = "Time",
ylab = "Threshold", vertex.size = "degree", vertex.color = NULL,
vertex.label = "", vertex.label.pos = NULL, vertex.label.cex = 1,
vertex.label.adj = c(0.5, 0.5), vertex.label.color = NULL,
vertex.sides = 40L, vertex.rot = 0, edge.width = 2,
edge.color = NULL, arrow.width = NULL, arrow.length = NULL,
arrows.color = NULL, include.grid = FALSE,
vertex.frame.color = NULL, bty = "n", jitter.factor = c(1, 1),
jitter.amount = c(0.25, 0.025), xlim = NULL, ylim = NULL,
edge.curved = NULL, background = NULL,
...)

Arguments

graph  A dynamic graph (see netdiffuseR-graphs).
expo  $n \times T$ matrix. Exposure to the innovation obtained from exposure
...  Additional arguments passed to plot.
toa  Integer vector of length $n$ with the times of adoption.
include_censored  Logical scalar. Passed to threshold.
t0  Integer scalar. Passed to threshold.
attrs  Passed to exposure (via threshold).
undirected  Logical scalar. When TRUE only the lower triangle of the adjacency matrix will
considered (faster).
no.contemporary  Logical scalar. When TRUE, edges for vertices with the same toa won’t be
plotted.
main  Character scalar. Title of the plot.
xlab  Character scalar. x-axis label.
ylab  Character scalar. y-axis label.
vertex.size  Numeric vector of size $n$. Relative size of the vertices.
vertex.color  Either a vector of size $n$ or a scalar indicating colors of the vertices.
vertex.label  Character vector of size $n$. Labels of the vertices.
vertex.label.pos  Integer value to be passed to text via pos.
vertex.label.cex  Either a numeric scalar or vector of size $n$. Passed to text.
vertex.label.adj  Passed to text.
vertex.label.color  Passed to text.
vertex.sides  Either a vector of size $n$ or a scalar indicating the number of sides of each vertex
(see details).
**vertex.rot**  Either a vector of size \( n \) or a scalar indicating the rotation in radians of each vertex (see details).

**edge.width**  Numeric. Width of the edges.

**edge.color**  Character. Color of the edges.

**arrow.width**  Numeric value to be passed to `arrows`.

**arrow.length**  Numeric value to be passed to `arrows`.

**arrow.color**  Color.

**include.grid**  Logical. When TRUE, the grid of the graph is drawn.

**vertex.frame.color**  Either a vector of size \( n \) or a scalar indicating colors of vertices' borders.

**bty**  See `par`.

**jitter.factor**  Numeric vector of size 2 (for x and y) passed to `jitter`.

**jitter.amount**  Numeric vector of size 2 (for x and y) passed to `jitter`.

**xlim**  Passed to `plot`.

**ylim**  Passed to `plot`.

**edge.curved**  Logical scalar. When curved, generates curved edges.

**background**  TBD

**Details**

When `vertex.label=NULL` the function uses vertices ids as labels. By default `vertex.label=""` plots no labels.

Vertices are drawn using an internal function for generating polygons. Polygons are inscribed in a circle of radius `vertex.size`, and can be rotated using `vertex.rot`. The number of sides of each polygon is set via `vertex.sides`.

**Author(s)**

George G. Vega Yon

**See Also**

Use `threshold` to retrieve the corresponding threshold obtained returned by `exposure`.

Other visualizations: `dgr`, `diffusionMap`, `drawColorKey`, `grid_distribution`, `hazard_rate`, `plot_adopters`, `plot_diffnet2`, `plot_diffnet`, `plot_infectsuscep`, `rescale_vertex_igraph`.

**Examples**

```r
# Generating a random graph
set.seed(1234)
n <- 6
nper <- 5
graph <- rgraph_er(n,nper, p=.3, undirected = FALSE)
toa <- sample(2000:(2000+nper-1), n, TRUE)
```
pretty_within

Pretty numbers within a range.

Description

A wrapper for `pretty`.

Usage

`pretty_within(x, min.n = 5, xrange = range(x, na.rm = TRUE), ...)`

Arguments

- **x**  
  Numeric vector passed to `pretty`.
- **min.n**  
  Integer scalar passed to `pretty`.
- **xrange**  
  Numeric vector of length 2. Indicates the range in which the output vector should lie on.
- **...**  
  Further arguments passed to the method.

The only difference with `pretty` is that this function subsets the resulting vector as

\[
\text{tick[(tick} \geq \text{xrange}[1]) \& (\text{tick} \leq \text{xrange}[2])]\]

Examples

```r
# Simple example
set.seed(3331)
x <- runif(10)
pretty(x)
pretty_within(x)
range(x)
```
Description

Simulates a diffusion network by creating a random dynamic network and adoption threshold levels.

Usage

```r
diffnet_multiple(R, statistic, ..., ncpus = 1L, cl = NULL)
diffnet(n, t, seed.nodes = "random", seed.p.adopt = 0.05,
        seed.graph = "scale-free", rgraph.args = list(), rewire = TRUE,
        rewire.args = list(), threshold.dist = runif(n),
        exposure.args = list(), name = "A diffusion network",
        behavior = "Random contagion", stop.no.diff = TRUE)
```

Arguments

- **R**: Integer scalar. Number of simulations to be done.
- **statistic**: A Function to be applied to each simulated diffusion network.
- **...**: Further arguments to be passed to `diffnet`.
- **ncpus**: Integer scalar. Number of processors to be used (see details).
- **cl**: An object of class `c("OCKcluster", "cluster")` (see details).
- **n**: Integer scalar. Number of vertices.
- **t**: Integer scalar. Time length.
- **seed.nodes**: Either a character scalar or a vector. Type of seed nodes (see details).
- **seed.p.adopt**: Numeric scalar. Proportion of early adopters.
- **seed.graph**: Baseline graph used for the simulation (see details).
- **rgraph.args**: List. Arguments to be passed to `rgraph`.
- **rewire**: Logical scalar. When TRUE, network slices are generated by rewiring (see `rewire_graph`).
- **rewire.args**: List. Arguments to be passed to `rewire_graph`.
- **threshold.dist**: Either a function to be applied via `sapply`, a numeric scalar, or a vector/matrix with `n` elements. Sets the adoption threshold for each node.
- **exposure.args**: List. Arguments to be passed to `exposure`.
- **name**: Character scalar. Passed to `as_diffnet`.
- **behavior**: Character scalar. Passed to `as_diffnet`.
- **stop.no.diff**: Logical scalar. When TRUE, the function will return with error if there was no diffusion. Otherwise it throws a warning.
Details

Instead of randomizing whether an individual adopts the innovation or not, this toy model randomizes threshold levels, seed adopters and network structure, so an individual adopts the innovation in time $T$ if his exposure is above or equal to his threshold. The simulation is done in the following steps:

1. Using `seed.graph`, a baseline graph is created.
2. Given the baseline graph, the set of initial adopters is defined using `seed.nodes`.
3. Afterwards, if `rewire=TRUE` $t - 1$ slices of the network are created by iteratively rewiring the baseline graph.
4. The `threshold.dist` function is applied to each node in the graph.
5. Simulation starts at $t = 2$ assigning adopters in each time period accordingly to each vertex’s threshold and exposure.

When `seed.nodes` is a character scalar it can be "marginal", "central" or "random". So each of these values sets the initial adopters using the vertices with lowest degree, with highest degree or completely randomly. The number of early adopters is set as `seed.p.adopt * n`. Please note that when marginal nodes are set as seed it may be the case that no diffusion process is attained as the chosen set of first adopters can be isolated. Any other case will be considered as an index (via `[<-methods`), hence the user can manually set the set of initial adopters, for example if the user sets `seed.nodes=c(1, 4, 7)` then nodes 1, 4 and 7 will be selected as initial adopters.

The argument `seed.graph` can be either a function that generates a graph (Any class of accepted graph format (see `netdiffuser-graphs`)), a graph itself or a character scalar in which the user sets the algorithm used to generate the first network (network in $t=1$), this can be either "scale-free" (Barabasi-Albert model using the `rgraph.ba` function, the default), "bernoulli" (Erdos-Renyi model using the `rgraph.er` function), or "small-world" (Watts-Strogatz model using the `rgraph.ws` function). The list `rgraph.args` passes arguments to the chosen algorithm.

When `rewire=TRUE`, the networks that follow $t=1$ will be generated using the `rewire.graph` function as $G(t) = R(G(t-1))$, where $R$ is the rewiring algorithm.

If a function, the argument `threshold.dist` sets the threshold for each vertex in the graph. It is applied using `sapply` as follows

\[
\text{sapply}(1:n, \text{threshold.dist})
\]

By default sets the threshold to be random for each node in the graph.

If `seed.graph` is provided, no random graph is generated and the simulation is applied using that graph instead.

`rewire.args` has the following default options:

- `p`: `.1`
  - `undirected`: `getOption("diffnet.undirected", FALSE)`
  - `self`: `getOption("diffnet.self", FALSE)`

`exposure.args` has the following default options:
The function `rdiffnet_multiple` is a wrapper of `rdiffnet` which allows simulating multiple diffusion networks with the same parameters and apply the same function to all of them. This function is designed to allow the user to perform larger simulation studies in which the distribution of a particular statistic is observed.

When `cl` is provided, then simulations are done via `parsapply`. If `ncpus` is greater than 1, then the function creates a cluster via `makeCluster` which is stopped (removed) once the process is complete.

**Value**

A random `diffnet` class object.

`rdiffnet_multiple` returns either a vector or an array depending on what statistic is (see `sapply` and `parsapply`).

**Author(s)**

George G. Vega Yon

**See Also**

Other simulation functions: `permute_graph`, `rewire_graph`, `rgraph_ba`, `rgraph_er`, `rgraph_ws`, `ring_lattice`

**Examples**

```r
# A simple example ---------------------------------------------------------------
set.seed(123)
z <- rdiffnet(100, 10)
z
summary(z)

# A more complex example: Adopt if at least one neighbor has adopted ----------
y <- rdiffnet(100, 10, threshold.dist=Function(x) 1,
exposure.args=list(valued=FALSE, normalized=FALSE))

# Re thinking the Adoption of Tetracycline -------------------------------------
newMI <- rdiffnet(seed.graph = medInnovationsDiffNet$graph,
     threshold.dist = threshold(medInnovationsDiffNet), rewire=FALSE)

# Simulation study comparing the diffusion with diff sets of seed nodes ------

# Random seed nodes
set.seed(1)
an="random", stop.no.diff=FALSE)
```
# Central seed nodes
set.seed(1)
ans1 <- rdifffnet_multiple(R=50, statistic=function(x) sum(!is.na(x$toa)),
n = 100, t = 4, seed.nodes = "central", stop.no.diff=FALSE)

boxplot(cbind(Random = ans0, Central = ans1), main="Number of adopters")

---

### Description

Reading pajek and Ucinet files, this function returns weighted edgelists in the form of data frames including a data frame of the vertices. (function on development)

### Usage

```r
read_pajek(x)
read_ml(x)
```

### Arguments

- **x** Character scalar. Path to the file to be imported.

### Details

Since .net files allow working with multi-relational networks (more than one class of edge), the function returns lists of edges and edgelist with the corresponding tag on the .net file. For example, if the .net file contains

```plaintext
*Arclist :9 "SAMPPR"
...  
*Arclist :10 "SAM NPR"
```

The output will include data frames of edgelists with those tags.

### Value

In the case of `read_pajek`, a list with three elements

- **vertices** A data frame with \( n \) rows and two columns: id and label
- **edges** If not null, a list of data frames with three columns: ego, alter, w (weight)
- **edgelist** If not null, a list of data frame with three columns: ego, alter, w (weight)

For `read_ml`, a list with two elements:

- **adjmat** An array with the graph
- **meta** A list with metadata

---

`read_pajek` *Read foreign graph formats*
read_ucinet_head

Author(s)

George G. Vega Yon

Source

From the pajek manual http://mrvar.fdv.uni-lj.si/pajek/pajekman.pdf

See Also

Other Foreign: igraph, network, read_ucinet_head

Examples

# From .net: Sampson monastery data from UCINET dataset ------------------------

# Reading the arcs/edges format
path <- system.file("extdata", "SAMPSON.NET", package = "netdiffuseR")
SAMPSON <- read_pajek(path)

# Reading the arcslist/edgelist format
path <- system.file("extdata", "SAMPSON.NET", package = "netdiffuseR")
SAMPSONL <- read_pajek(path)

# From DL (UCINET): Sampson monastery data (again) ----------------------------
path <- system.file("extdata", "SAMPSON.DAT", package = "netdiffuseR")
SAMPSONL <- read_ml(path)

read_ucinet_head

Reads UCINET files

Description

Reads UCINET files

Read UCINET files (binary)

Usage

read_ucinet_head(f)

read_ucinet(f, echo = FALSE)

Arguments

f Character scalar. Name of the header file. e.g. mydata.##h.

echo Logical scalar. When TRUE shows a message.
Recode an edgelist such that ids go from 1 to n.

Recode an edgelist such that ids go from 1 to n.

Recode an edgelist such that ids go from 1 to n.

Recode an edgelist such that ids go from 1 to n.

Recode an edgelist such that ids go from 1 to n.
Value

A recoded edgelist as a two-column matrix/data.frame depending on the class of data. The output includes an attribute called "recode" which contains a two column data.frame providing a mapping between the previous code and the new code (see the examples).

Author(s)

George G. Vega Yon

See Also

edgelist_to_adjmat

Examples

# Simple example ---------------------------------
edgelist <- cbind(c(1,1,3,6),c(4,3,200,1))
edgelist
recoded_edgelist <- recode(edgelist)
recoded_edgelist

# Retrieving the "recode" attribute
attr(recoded_edgelist, "recode")

rescale_vertex_igraph  Rescale vertex size to be used in plot.igraph.

Description

This function rescales a vertex size before passing it to plot.igraph so that the resulting vertices have the desired size relative to the x-axis.

Usage

rescale_vertex_igraph(vertex.size, par.usr = par("usr"),
                       minmax.relative.size = getOption("diffnet.minmax.relative.size",
                       c(0.01, 0.04)), adjust = 200)

igraph_vertex_rescale(vertex.size, par.usr = par("usr"),
                       minmax.relative.size = getOption("diffnet.minmax.relative.size",
                       c(0.01, 0.04)), adjust = 200)

vertex_rescale_igraph(vertex.size, par.usr = par("usr"),
                       minmax.relative.size = getOption("diffnet.minmax.relative.size",
                       c(0.01, 0.04)), adjust = 200)
Arguments

vertex.size: Numeric vector of unscaled vertices’ sizes. This is unit-free.

par.usr: Integer vector of length 4 with the coordinates of plotting region. By default uses par("usr").

minmax.relative.size: A numeric vector of length 2. Represents the desired min and max vertex sizes relative to the x-axis in terms of percentage (see details).

adjust: Numeric scalar. Adjustment made to the resulting adjusted size (see details).

Details

minmax.relative.size limits the minimum and maximum size that a vertex can take in the plot relative to the x-axis scale. The values for the x-axis scale are by default retrieved by accessing to par("usr"). By default the vertex are rescaled to be at least 1% of the size of the plotting region and no more than 5% of the plotting region, minmax.relative.size=c(0.01, 0.05).

The default value for adjust is taken from igraph version 1.0.1. In particular, the function igraph::igraph.shape.circle.plot, in which before passing the vertex.size to the function symbols, the vertex size is reduced by 200.

The rescaling is as follows:

\[ v' = \frac{v - v}{\bar{v} - v} \times (\bar{s} - s) + \bar{s} \]

Where \( v \) is the vertex size, \( \bar{v} \) and \( \bar{v} \) are the max and min values of \( v \) respectively, and \( \bar{s} \) and \( \bar{s} \) are the max and min size that vertices take in terms of minmax.relative.size and par.usr. The adjusted value \( v' \) is then multiplied by adjust.

igraph_vertex_rescale and vertex_rescale_igraph are aliases.

Value

An integer vector of the same length as vertex.size with rescaled values.

Author(s)

George G. Vega Yon

See Also

Other visualizations: dgr, diffusionMap, drawColorKey, grid_distribution, hazard_rate, plot_adopters, plot_diffnet2, plot_diffnet, plot_infectsuscep, plot_threshold

Examples

library(igraph)

# Random graph and coordinates
set.seed(2134)
g <- barabasi.game(10)
rewire_graph

coords <- layout_nicely(g)

# Random size and figures
size <- runif(10)
size <- cbind(size, size)
shap <- sample(c("circle", "square"), 10, TRUE)

# Plotting
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2), mai=rep(.5,4))
for (i in seq(1, 1000, length.out = 4)) {
  # New plot-window
  plot.new()
  plot.window(xlim=range(coords[,1]*i), ylim=range(coords[,2]*i))

  # plotting graph
  plot(g, layout=coords*i, add=TRUE, rescale=FALSE,
       vertex.shape = shap,
       vertex.size = rescale_vertex_igraph(size) # HERE WE RESCALE!
  )

  # Adding some axis
  axis(1, lwd=0, lwd.ticks = 1)
  axis(2, lwd=0, lwd.ticks = 1)
  box()
}
par(oldpar)

---

**Description**

Changes the structure of a graph by altering ties.

**Usage**

```r
rewire_graph(graph, p, algorithm = "endpoints", both.ends = FALSE,
              self = FALSE, multiple = FALSE,
              undirected = getOption("diffnet.undirected"),
              pr.change = ifelse(self, 0.5, 1), copy.first = TRUE,
              altheagons = FALSE)
```

**Arguments**

- **graph**: Any class of accepted graph format (see `netdiffuserR-graphs`).
- **p**: Either a [0,1] vector with rewiring probabilities (algorithm="endpoints"), or an integer vector with number of iterations (algorithm="swap").
algorithm
both.ends
self
multiple
undirected
pr.change
copy.first
althexagons

Character scalar. Either "swap", "endpoints", or "qap" (see rewire_qap).
Logical scalar. When TRUE rewires both ends.
Logical scalar. When TRUE, allows loops (self edges).
Logical scalar. When TRUE allows multiple edges.
Logical scalar. When TRUE only the lower triangle of the adjacency matrix will considered (faster).
Numeric scalar. Probability ([0,1]) of doing a rewire (see details).
Logical scalar. When TRUE and graph is dynamic uses the first slice as a baseline for the rest of slices (see details).
Logical scalar. When TRUE uses the compact alternating hexagons algorithm (currently ignored [on development]).

Details

The algorithm "qap" is described in rewire_qap, and only uses graph from the arguments (since it is simply relabelling the graph).

In the case of "swap" and "endpoints", both algorithms are implemented sequentially, this is, edge-wise checking self edges and multiple edges over the changing graph; in other words, at step $m$ (in which either a new endpoint or edge is chosen, depending on the algorithm), the algorithms verify whether the proposed change creates either multiple edges or self edges using the resulting graph at step $m - 1$.

The main difference between the two algorithms is that the "swap" algorithm preserves the degree sequence of the graph and "endpoints" does not. The "swap" algorithm is specially useful to asses the non-randomness of a graph’s structural properties, furthermore it is this algorithm the one used in the struct_test routine implemented in netdiffuseR.

Rewiring assumes a weighted network, hence $G(i,j) = k = G(i',j')$, where $i', j'$ are the new end points of the edge and $k$ may not be equal to one.

In the case of dynamic graphs, when copy.first=TRUE, after rewiring the first slice–$t = 1$–the rest of slices are generated by rewiring the rewired version of the first slice. Formally:

$$ G(t)' = \begin{cases} 
R(G(t)) & \text{if } t = 1 \\
R(G(1)') & \text{otherwise}
\end{cases} $$

Where $G(t)$ is the t-th slice, $G(t)'$ is the t-th rewired slice, and $R$ is the rewiring function. Otherwise, copy.first=FALSE (default), The rewiring function is simply $G(t)' = R(G(t))$.

The following sections describe the way both algorithms were implemented.

Swap algorithm

The "swap" algorithm chooses randomly two edges $(a, b)$ and $(c, d)$ and swaps the 'right' endpoint of boths such that we get $(a, d)$ and $(c, b)$ (considering self and multiple edges).

Following Milo et al. (2004) testing procedure, the algorithm shows to be well behaved in terms of been unbiased, so after each iteration each possible structure of the graph has the same probability of been generated. The algorithm has been implemented as follows:

Let $E$ be the set of edges of the graph $G$. For $i = 1$ to $p$, do:
1. With probability 1-pr \cdot change got to the last step.
2. Choose \( e_0 = (a, b) \) from \( E \). If \(!self \ & \ a == b \) then go to the last step.
3. Choose \( e_1 = (c, d) \) from \( E \). If \(!self \ & \ c == d \) then go to the last step.
4. Define \( e_0' = (a, d) \) and \( e_1' = (c, b) \). If \(!multiple \ & \ G[e_0'] != 0 \ | \ G[e_1'] != 0 \) then go to the last step.(*)
5. Define \( v_0 = G[e_0] \) and \( v_1 = G[e_1] \), set \( G[e_0] = 0 \) and \( G[e_1] = 0 \) (and the same to the diagonally opposed coordinates in the case of undirected graphs)
6. Set \( G[e_0'] = v_0 \) and \( G[e_1'] = v_1 \) (and so with the diagonally opposed coordinates in the case of undirected graphs).
7. Next \( l \).

(*) When altheagons=TRUE, the algorithm changes and applies what Rao et al. (1996) describe as Compact Alternating Hexagons. This modification assures the algorithm to be able to achieve any structure. The algorithm consists on doing the following swapping: \((i1i2, i1i3, i2i3, i2i1, i3i1, i3i2)\) with values \((1, 0, 1, 0, 1, 0)\) respectively with \( i1! = i2! = i3 \). See the examples and references.

In Milo et al. (2004) is suggested that in order for the rewired graph to be independent from the original one researchers usually iterate around \( nlinks(graph) \* 100 \) times, so \( p=nlinks(graph) \* 100 \). On the other hand in Ray et al (2012) it is shown that in order to achive such it is needed to perform \( nlinks(graph) \* \log(1/\text{eps}) \), where \( \text{eps} \approx 1 \times 10^{-7} \), in other words, around \( nlinks(graph) \* 16 \). We set the default to be 20.

In the case of Markov chains, the variable pr\cdot change allows making the algorithm aperiodic. This is relevant only if the probability self-loop to a particular state is null, for example, if we set \( self=True \) and \( multiple=True \), then in every step the algorithm will be able to change the state. For more details see Stanton and Pinar (2012) [p. 3.5:9].

**Endpoints algorithm**

This reconnect either one or both of the endpoints of the edge randomly. As a big difference with the swap algorithm is that this does not preserves the degree sequence of the graph (at most the outgoing degree sequence). The algorithm is implemented as follows:

Let \( G \) be the baseline graph and \( G' \) be a copy of it. Then, For \( l = 1 \) to \(|E|\) do:

1. Pick the \( l \)-th edge from \( E \), define it as \( e = (i, j) \).
2. Draw \( r \) from \( U(0, 1) \), if \( r > p \) go to the last step.
3. If \(!\text{undirected} \ & \ i < j \) go to the last step.
4. Randomly select a vertex \( j' \) (and \( i' \) if both\_ends==TRUE). And define \( e' = (i, j') \) (or \( e' = (i', j') \) if both\_ends==TRUE).
5. If \(!self \ & \ i==j' \) (or if both\_ends==TRUE \ & \ i'==j) go to the last step.
6. If \(!\text{multiple} \ & \ G'[e']!=0 \) then go to the last step.
7. Define \( v = G[e] \), set \( G'[e] = 0 \) and \( G'[e'] = v \) (and the same to the diagonally opposed coordinates in the case of undirected graphs).
8. Next \( l \).

The endpoints algorithm is used by default in rdiffnet and used to be the default in struct_test (now swap is the default).
 rewire_graph

Author(s)
George G. Vega Yon

References


See Also
Other simulation functions: permute_graph, rdiffnet, rgraph_ba, rgraph_er, rgraph_ws, ring_lattice

Examples
# Checking the consistency of the "swap" ----------------------------------------

# A graph with known structure (see Milo 2004)
N <- 5
x <- matrix(0, ncol=N, nrow=N)
x <- as(x, "dgCMatrix")
x[1,c(-1,-N)] <- 1
x[c(-1,-N),N] <- 1
x

# Simulations (increase the number for more precision)
set.seed(8612)
nsim <- 1e4
w <- sapply(seq_len(nsim), function(y) {
  # Creating the new graph
g <- rewire_graph(x,p=nlinks(x)*100, algorithm = "swap")

  # Categorizing (tag of the generated structure)
paste0(as.vector(g), collapse="")}
})
# Counting
coded <- as.integer(as.factor(w))

plot(table(coded)/nsim*100, type="p", ylab="Frequency %", xlab="Class of graph", pch=3,
main="Distribution of classes generated by rewiring")

# Marking the original structure
baseline <- paste0(as.vector(x), collapse="")
points(x=7,y=table(as.factor(w))[baseline]/nsim*100, pch=3, col="red")

rgraph_ba  Scale-free and Homophilic Random Networks

Description

Generates a scale-free random graph based on Bollabas et al. (2001), also know as Linearized Chord Diagram (LCD) which has nice mathematical propoerties. And also scale-free homophilic networks when an vertex attribute \( \eta \) is passed.

Usage

\[
\text{rgraph\_ba}(m0 = 1L, m = 1L, t = 10L, graph = \text{NULL}, \text{self} = \text{TRUE}, \\
\text{eta} = \text{NULL})
\]

Arguments

- \( m0 \): Integer scalar. Number of initial vertices in the graph.
- \( m \): Integer scalar. Number of new edges per vertex added.
- \( t \): Integer scalar. Number of time periods (steps).
- \( graph \): Any class of accepted graph format (see netdiffuser\_graphs).
- \( self \): Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see de- tails).
- \( eta \): Numeric vector of length \( t + m0 \). When specified, it generates a scale-free homophilic network (see details).

Details

Based on Ballobás et al. (2001) creates a directed random graph of size \( t + m0 \). A big difference with B-A model is that this allows for loops (self/auto edges) and further multiple links, nevertheless, as \( t \) increases, the number of such cases reduces.

By default, the degree of the first \( m0 \) vertices is set to be 2 (loops). When \( m > 1 \), as described in the paper, each new link from the new vertex is added one at a time "counting ‘outward half’ of the edge being added as already contributing to the degrees".
When `self=FALSE`, the generated graph is created without autolinks. This means that at the beginning, if the number of links equals zero, all vertices have the same probability of receiving a new link.

When `eta` is passed, it implements the model specified in De Almeida et al. (2013), a scale-free homophilic network. To do so `eta` is rescaled to be between 0 and 1 and the probability that the node `i` links to node `j` is as follows:

\[
\frac{(1 - A_{ij})k_j}{\sum_j (1 - A_{ij})k_j}
\]

Where \(A_{ij} = |\eta_i - \eta_j|\) and \(k_j\) is the degree of the \(j\)-th vertex.

**Value**

If `graph` is not provided, a static graph, otherwise an expanded graph (additional vertices) of the same class as `graph`.

The resulting graph will have `graph$meta$undirected = FALSE` if it is of class `diffnet` and `attr(graph, "undirected")=FALSE` otherwise.

**Author(s)**

George G. Vega Yon

**References**


**See Also**

Other simulation functions: `permute_graph`, `rdiffnet`, `rewire_graph`, `rgraph_er`, `rgraph_ws`, `ring_lattice`

**Examples**

# Using another graph as a base graph

```r
graph <- rgraph_ba()
graph

graph <- rgraph_ba(graph=graph)
```
rgraph_er

Erdoes-Renyi model

Description
Generates a bernoulli random graph.

Usage
rgraph_er(n = 10, t = 1, p = 0.01,
undirected = getOption("diffnet.undirected"), weighted = FALSE,
self = getOption("diffnet.self"), as.edgelist = FALSE)

Arguments

n
Integer. Number of vertices

t
Integer. Number of time periods

p
Double. Probability of a link between ego and alter.

undirected
Logical scalar. Whether the graph is undirected or not.

weighted
Logical. Whether the graph is weighted or not.

self
Logical. Whether it includes self-edges.

as.edgelist
Logical. When TRUE the graph is presented as an edgelist instead of an adjacency matrix.
Details

For each pair of nodes \( \{i, j\} \), an edge is created with probability \( p \), this is, \( \Pr\{Linki − j\} = Pr\{x < p\} \), where \( x \) is drawn from a \( Uniform(0,1) \).

When \texttt{weighted=TRUE}, the strength of ties is given by the random draw \( x \) used to compare against \( p \), hence, if \( x < p \) then the strength will be set to \( x \).

In the case of dynamic graphs, the algorithm is repeated \( t \) times, so the networks are uncorrelated.

Value

A graph represented by an adjacency matrix (if \( t=1 \)), or an array of adjacency matrices (if \( t>1 \)).

Note

The resulting adjacency matrix is store as a dense matrix, not as a sparse matrix, hence the user should be careful when choosing the size of the network.

Author(s)

George G. Vega Yon

References


See Also

Other simulation functions: \texttt{permute_graph, rdiffnet, rewire_graph, rgraph_ba, rgraph_ws, ring_lattice}

Examples

```
## Not run:
# Setting the seed
set.seed(123)

# Generating an directed graph
rgraph.er(undirected=FALSE)

# Comparing P(tie)
x <- rgraph.er(1000, p=.1)
sum(x)/length(x)

# Several period random gram
rgraph.er(t=5)

## End(Not run)
```
**rgraph_ws** *Watts-Strogatz model*

**Description**
Generates a small-world random graph.

**Usage**
```
rgraph_ws(n, k, p, both.ends = FALSE, self = FALSE, multiple = FALSE, undirected = FALSE)
```

**Arguments**
- **n** Integer scalar. Set the size of the graph.
- **k** Integer scalar. Set the initial degree of the ring (must be less than \( n \)).
- **p** Numeric scalar/vector of length \( T \). Set the probability of changing an edge.
- **both.ends** Logical scalar. When TRUE rewires both ends.
- **self** Logical scalar. When TRUE, allows loops (self edges).
- **multiple** Logical scalar. When TRUE allows multiple edges.
- **undirected** Logical scalar. Passed to `ring_lattice`

**Details**
Implemented as in Watts and Strogatz (1998). Starts from an undirected ring with \( n \) vertices all with degree \( k \) (so it must be an even number), and then rewire each edge by setting the endpoint (so now you treat it as a digraph) randomly any vertex in \( N \setminus i \) avoiding multiple links (by default) using the rewiring algorithm described on the paper.

**Value**
A random graph of size \( n \times n \) following the small-world model. The resulting graph will have `attr(graph, "undirected")=FALSE`.

**Author(s)**
George G. Vega Yon

**References**
See Also

Other simulation functions: permute_graph, rdifnet, rewire_graph, rgraph_ba, rgraph_er, ring_lattice

Examples

library(igraph)
set.seed(7123)
x0 <- graph_from_adjacency_matrix(rgraph_ws(10, 2, 0))
x1 <- graph_from_adjacency_matrix(rgraph_ws(10, 2, .3))
x2 <- graph_from_adjacency_matrix(rgraph_ws(10, 2, 1))
oldpar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(x0, layout=layout_in_circle, edge.curved=TRUE, main="Regular")
plot(x1, layout=layout_in_circle, edge.curved=TRUE, main="Small-world")
plot(x2, layout=layout_in_circle, edge.curved=TRUE, main="Random")
par(oldpar)

ring_lattice

Ring lattice graph

Description

Creates a ring lattice with \( n \) vertices, each one of degree (at most) \( k \) as an undirected graph. This is the basis of rgraph_ws.

Usage

ring_lattice(n, k, undirected = FALSE)

Arguments

\( n \) Integer scalar. Size of the graph.
\( k \) Integer scalar. Out-degree of each vertex.
undirected Logical scalar. Whether the graph is undirected or not.

Details

when undirected=TRUE, the degree of each node always even. So if \( k=3 \), then the degree will be 2.

Value

A sparse matrix of class dgCMatrix of size \( n \times n \).
**round_to_seq**

**References**


**See Also**

Other simulation functions: `permute_graph`, `rdiffnet`, `rewire_graph`, `rgraph.ba`, `rgraph.er`, `rgraph.ws`

---

**round_to_seq**

*Takes a numeric vector and maps it into a finite length sequence*

**Description**

Takes a numeric vector and maps it into a finite length sequence

**Usage**

```r
round_to_seq(x, nlevels = 20, as_factor = FALSE)
```

**Arguments**

- `x` A numeric or integer vector.
- `nlevels` Integer scalar. Length of the sequence to be map onto.
- `as_factor` Logical scalar. When TRUE the resulting vector is factor.

**Value**

A vector of length `length(x)` with values mapped to a sequence with `nlevels` unique values

**See Also**

Used in `diffmap` and `plot_diffnet`

**Examples**

```r
x <- rnorm(100)
w <- data.frame(as.integer(round_to_seq(x, as_factor = TRUE)),x)
plot(w,x)
```
select_egoalter  

*Calculate the number of adoption changes between ego and alter.*

**Description**

This function calculates the 16 possible configurations between ego and alter over two time points in terms of their behavior and tie changes. From time one to time two, given a binary state of behavior, ego and alter can be related in 16 different ways. The function `adopt_changes` is just an alias for `select_egoalter`.

**Usage**

```r
select_egoalter(graph, adopt, period = NULL)

adopt_changes(graph, adopt, period = NULL)
```

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

**Arguments**

- `graph` A dynamic graph (see `netdiffuseR-graphs`).
- `adopt` \(n \times T\) matrix. Cumulative adoption matrix obtained from `toa_mat`.
- `period` Integer scalar. Optional to make the count for a particular period of time.
- `object` An object of class `diffnet_adoptChanges`.
- `...` Ignored.

**Details**

The 16 possibilities are summarized in this matrix:

<table>
<thead>
<tr>
<th>Alter</th>
<th>(t - 1)</th>
<th>(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>16</td>
</tr>
</tbody>
</table>

The first two Yes/No columns represent Ego’s adoption of the innovation in \(t - 1\) and \(t\); while the first two Yes/No rows represent Alter’s adoption of the innovation in \(t - 1\) and \(t\) respectively. So for example, number 4 means that while neither of the two had adopted the innovation in \(t - 1\), both have in \(t\). At the same time, number 12 means that ego adopted the innovation in \(t\), but alter had already adopted in \(t - 1\) (so it has it in both, \(t\) and \(t - 1\)).
struct_equiv

Value
An object of class diffnet_adoptChanges and data.frame with \( n \times (T - 1) \) rows and \( 2 + 16 \times 3 \) columns. The column names are:

- **time**: Integer representing the time period
- **id**: Node id
- **select_a_01, ..., select_a_16**: Number of new links classified between categories 1 to 16.
- **select_d_01, ..., select_d_16**: Number of remove links classified between categories 1 to 16.
- **select_s_01, ..., select_s_16**: Number of unchanged links classified between categories 1 to 16.

Author(s)
George G. Vega Yon & Thomas W. Valente

References
Thomas W. Valente, Stephanie R. Dyal, Kar-Hai Chu, Heather Wipfli, Kayo Fujimoto, *Diffusion of innovations theory applied to global tobacco control treaty ratification*, Social Science & Medicine, Volume 145, November 2015, Pages 89-97, ISSN 0277-9536 (http://dx.doi.org/10.1016/j.socscimed.2015.10.001)

Examples

```r
# Simple example
set.seed(1312)
dn <- rdifnet(20, 5, seed.graph="small-world")
ans <- adopt_changes(dn)
str(ans)
summary(ans)
```

---

**struct_equiv**

*Structural Equivalence*

Description
Computes structural equivalence between ego and alter in a network

Usage

```r
struct_equiv(graph, v = 1, inf.replace = 0, groupvar = NULL, ...)
```

# S3 method for class 'diffnet_se'
print(x, ...)

Arguments

- **graph**: Any class of accepted graph format (see `netdiffuserR-graphs`).
- **v**: Numeric scalar. Cohesion constant (see details).
- **inf.replace**: Deprecated.
- **groupvar**: Either a character scalar (if `graph` is `diffnet`), or a vector of size `n`. Further arguments to be passed to `approx_geodesic` (not valid for the print method).
- **x**: A `diffnet_se` class object.

Details

Structure equivalence is computed as presented in Valente (1995), and Burt (1987), in particular

\[ SE_{ij} = \frac{(d_{max} - d_{ji})^v}{\sum_{k \neq i} (d_{max} - d_{ki})^v} \]

with the summation over \( k \neq i \), and \( d_{ji} \), Euclidean distance in terms of geodesics, is defined as

\[ d_{ji} = \left[ (z_{ji} - z_{ij})^2 + \sum_k (z_{jk} - z_{ik})^2 + \sum_k (z_{ki} - z_{kj})^2 \right]^{\frac{1}{2}} \]

with \( z_{ij} \) as the geodesic (shortest path) from \( i \) to \( j \), and \( d_{max} \), equal to largest Euclidean distance between \( i \) and any other vertex in the network. All summations are made over \( k \notin \{i,j\} \).

Here, the value of \( v \) is interpreted as cohesion level. The higher its value, the higher will be the influence that the closest alters will have over ego (see Burt’s paper in the reference).

Structural equivalence can be computed either for the entire graph or by groups of vertices. When, for example, the user knows beforehand that the vertices are distributed across separated communities, he can make this explicit to the function and provide a `groupvar` variable that accounts for this. Hence, when `groupvar` is not NULL the algorithm will compute structural equivalence within communities as marked by `groupvar`.

Value

If `graph` is a static graph, a list with the following elements:

- **SE**: Matrix of size \( n \times n \) with Structural equivalence
- **d**: Matrix of size \( n \times n \) Euclidean distances
- **gdist**: Matrix of size \( n \times n \) Normalized geodesic distance

In the case of dynamic graph, is a list of size \( t \) in which each element contains a list as described before. When `groupvar` is specified, the resulting matrices will be of class `dgCMatrix`, otherwise will be of class `matrix`.

Author(s)

George G. Vega Yon & Thomas W. Valente
References


See Also

Other statistics: bass, classify_adopters, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, infection, moran, threshold, vertex_covariate_dist

Examples

# Computing structural equivalence for the fakedata --------------------------
data(fakesurvey)

# Coercing it into a diffnet object
fakediffnet <- survey_to_diffnet(
  fakesurvey, "id", c("net1", "net2", "net3"), "toa", "group"
)

# Computing structural equivalence without specifying group
se_all <- struct_equiv(fakediffnet)

# Notice that pairs of individuals from different communities have
# non-zero values
se_all
se_all[[1]]$SE

# ... Now specifying a groupvar
se_group <- struct_equiv(fakediffnet, groupvar="group")

# Notice that pairs of individuals from different communities have
# only zero values.
se_group
se_group[[1]]$SE

---

**struct_test**

*Structure dependence test*

**Description**

Test whether or not a network estimates can be considered structurally dependent, i.e. a function of the network structure. By rewiring the graph and calculating a particular statistic \( t \), the test compares the observed mean of \( t \) against the empirical distribution of it obtained from rewiring the network.
Usage

n_rewires(graph, p = c(20L, rep(0.1, nslices(graph) - 1)))

struct_test(graph, statistic, R, rewire.args = list(), ...)

## S3 method for class 'diffnet_struct_test'
c(..., recursive = FALSE)

## S3 method for class 'diffnet_struct_test'
print(x, ...)

## S3 method for class 'diffnet_struct_test'
hist(x,
  main = "Empirical Distribution of Statistic",
  xlab = expression(Values ~ of ~ t), breaks = 20, annotated = TRUE,
  b0 = expression(atop(plain("") %up% plain("")), t[0]),
  b = expression(atop(plain("") %up% plain("")), t[]), ask = TRUE,
  ...)

struct_test_asymp(graph, Y, statistic_name = "distance", p = 2, ...)

Arguments

graph A diffnet graph.
p Either a Numeric scalar or vector of length nslices(graph)~1 with the number of rewires per links.
statistic A function that returns either a scalar or a vector.
R Integer scalar. Number of repetitions.
rewire.args List. Arguments to be passed to rewire_graph
... Further arguments passed to the method (see details).
recursive Ignored
x A diffnet_struct_test class object.
main Character scalar. Title of the histogram.
xlab Character scalar. x-axis label.
breaks Passed to hist.
annotated Logical scalar. When TRUE marks the observed data average and the simulated data average.
b0 Character scalar. When annotated=TRUE, label for the value of b0.
b Character scalar. When annotated=TRUE, label for the value of b.
ask Logical scalar. When TRUE, asks the user to type <Enter> to see each plot (as many as statistics where computed).
Y Numeric vector of length n.
statistic_name Character scalar. Name of the metric to compute. Currently this can be either "distance", ">", ",<", ",=", ",=>", or ",<=".
Details

struct_test computes the test by generating the null distribution using Monte Carlo simulations (rewiring). struct_test_asym computes the test using an asymptotic approximation. While available, we do not recommend using the asymptotic approximation since it has not shown good results when compared to the MC approximation. Furthermore, the asymptotic version has only been implemented for graph as static graph.

The output from the hist method is the same as hist.default.

struct_test is a wrapper for the function boot from the boot package. Instead of resampling data–vertices or edges–in each iteration the function rewires the original graph using rewire_graph and applies the function defined by the user in statistic.

The default values to rewire_graph via rewire.args are:

- \( p \) Number or Integer with default n.rewires(graph).
- undirected Logical scalar with default getOption("diffnet.undirected", FALSE).
- copy.first Logical scalar with TRUE.
- algorithm Character scalar with default "swap".

In struct_test ... are passed to boot, otherwise are passed to the corresponding method (hist for instance).

From the print method, p-value for the null of the statistic been equal between graph and its rewired versions is computed as follows

\[
p(\tau) = 2 \times \min \left( \Pr(t \leq \tau), \Pr(t \geq \tau) \right)
\]

Where \( \Pr(\cdot) \) is approximated using the Empirical Distribution Function retrieved from the simulations.

For the case of the asymptotic approximation, under the null we have

\[
\sqrt{n} \left( \hat{\beta}(Y, G) - \mu_\beta \right) \overset{d}{\sim} N \left( 0, \sigma_\beta^2 \right)
\]

The test is actually on development by Vega Yon and Valente. A copy of the working paper can be distributed upon request to <g.vegayon@gmail.com>.

The function n.rewires proposes a vector of number of rewirings that are performed in each iteration.

Value

A list of class diffnet_struct_test containing the following:

- graph The graph passed to struct_test.
- p.value The resulting p-value of the test (see details).
- t0 The observed value of the statistic.
- mean_t The average value of the statistic applied to the simulated networks.
- R Number of simulations.
statistic  The function statistic passed to struct_test.
boot     A boot class object as return from the call to boot.
rewire.args  The list rewire.args passed to struct_test.

Author(s)
George G. Vega Yon

References
University Press.

See Also
Other Functions for inference: bootnet, moran

Examples

# Creating a random graph
set.seed(881)
diffnet <- rdifffnet(100, 5, seed.graph="small-world")

# Testing structure-dependency of threshold
res <- struct_test(diffnet, function(g) mean(threshold(g), na.rm=TRUE), R=100)
res
hist(res)

# Adding a legend
legend("topleft", bty="n",
legend=c(
  expression(t[0]:Baseline),
  expression(t:-Rewired-average)
)
)

# Concatenating results
c(res, res)

# Running in parallel fashion
## Not run:
res <- struct_test(diffnet, function(g) mean(threshold(g), na.rm=TRUE), R=100,
ncpus=4, parallel="multicore")
res
hist(res)
## End(Not run)
Summary of diffnet objects

Description

Summary of diffnet objects

Usage

## S3 method for class 'diffnet'
summary(object, slices = NULL, no.print = FALSE,
skip.moran = FALSE, valued = getOption("diffnet.valued", FALSE), ...)

Arguments

- **object**: An object of class `diffnet`.
- **slices**: Either an integer or character vector. While integer vectors are used as indexes, character vectors are used jointly with the time period labels.
- **no.print**: Logical scalar. When TRUE suppress screen messages.
- **skip.moran**: Logical scalar. When TRUE Moran’s I is not reported (see details).
- **valued**: Logical scalar. When TRUE weights will be considered. Otherwise non-zero values will be replaced by ones.
- ... Further arguments to be passed to `approx_geodesic`.

Details

Moran’s I is calculated over the cumulative adoption matrix using as weighting matrix the inverse of the geodesic distance matrix. All this via `moran`. For each time period $t$, this is calculated as:

$$ m = \text{moran}(C[], t, G^\neq(-1)) $$

Where $C[][t]$ is the $t$-th column of the cumulative adoption matrix, $G\neq(-1)$ is the element-wise inverse of the geodesic matrix at time $t$, and `moran` is `netdiffuseR`'s moran’s I routine. When `skip.moran=TRUE` Moran’s I is not reported. This can be useful for both: reducing computing time and saving memory as geodesic distance matrix can become large. Since version 1.18.6, geodesic matrices are approximated using `approx_geodesic` which, as a difference from `geodist` from the `sna` package, and `distances` from the `igraph` package returns a matrix of class `dgCMatrix` (more details in `approx_geodesic`).

Value

A data frame with the following columns:

- **adopt**: Integer. Number of adopters at each time point.
- **cum_adopt**: Integer. Number of cumulative adopters at each time point.
cum_adopt_pcent
   Numeric. Proportion of cumulative adopters at each time point.

hazard
   Numeric. Hazard rate at each time point.

density
   Numeric. Density of the network at each time point.

moran_obs
   Numeric. Observed Moran’s I.

moran_exp
   Numeric. Expected Moran’s I.

moran_sd
   Numeric. Standard error of Moran’s I under the null.

moran_pval
   Numeric. P-value for the observed Moran’s I.

Author(s)

George G. Vega Yon

See Also

Other diffnet methods: \%\% as.array.diffnet, c.diffnet, diffnet-arithmetic, diffnet-class,
diffnet_index, plot.diffnet

Examples

data(medInnovationsDiffNet)
summary(medInnovationsDiffNet)

Description

These convenient functions turn network nomination datasets and edgelists with vertex attributes
datasets into diffnet objects. Both work as wrappers of edgelist_to_adjmat and new_diffnet.

Usage

survey_to_diffnet(dat, idvar, netvars, toavar, groupvar = NULL,
   no.unsurveyed = TRUE, timevar = NULL, t = NULL,
   undirected = getOption("diffnet.undirected", FALSE),
   self = getOption("diffnet.self", FALSE),
   multiple = getOption("diffnet.multiple", FALSE),
   keep.isolates = TRUE, recode.ids = TRUE, warn.coercion = TRUE, ...)

edgelist_to_diffnet(edgelist, w = NULL, t0 = NULL, t1 = NULL, dat,
   idvar, toavar, timevar = NULL,
   undirected = getOption("diffnet.undirected", FALSE),
   self = getOption("diffnet.self", FALSE),
   multiple = getOption("diffnet.multiple", FALSE), fill.missing = NULL,
   keep.isolates = TRUE, recode.ids = TRUE, warn.coercion = TRUE)
survey_to_diffnet

Arguments

- **dat**: A data frame.
- **idvar**: Character scalar. Name of the id variable.
- **netvars**: Character vector. Names of the network nomination variables.
- **toavar**: Character scalar. Name of the time of adoption variable.
- **groupvar**: Character scalar. Name of cohort variable (e.g. city).
- **no.unsurveyed**: Logical scalar. When TRUE the nominated individuals that do not show in idvar are set to NA (see details).
- **timevar**: Character scalar. In the case of longitudinal data, name of the time var.
- **t**: Integer scalar. Repeat the network t times (if no t0, t1 are provided).
- **undirected**: Logical scalar. When TRUE only the lower triangle of the adjacency matrix will be considered (faster).
- **self**: Logical scalar. When TRUE autolinks (loops, self edges) are allowed (see details).
- **multiple**: Logical scalar. When TRUE allows multiple edges.
- **keep.isolates**: Logical scalar. When FALSE, rows with NA,NULL values (isolated vertices unless have autolink) will be dropped (see details).
- **recode.ids**: Logical scalar. When TRUE ids are recoded using as.factor (see details).
- **warn.coercion**: Logical scalar. When TRUE warns coercion from numeric to integer.
- **edgelist**: Two column matrix/data.frame in the form of ego-source- and alter-target- (see details).
- **w**: Numeric vector. Strength of ties (optional).
- **t0**: Integer vector. Starting time of the ties (optional).
- **t1**: Integer vector. Finishing time of the ties (optional).
- **fill.missing**: Character scalar. In the case of having unmatching ids between dat and edgelist, fills the data (see details).

Details

All of netvars, toavar and groupvar must be integers. Were these numeric they are coerced into integers, otherwise, when neither of both, the function returns with error. idvar, on the other hand, should only be integer when calling survey_to_diffnet, on the contrary, for edgelist_to_diffnet, idvar may be character.

In field work it is not unusual that some respondents nominate unsurveyed individuals. In such case, in order to exclude them from the analysis, the user can set no.unsurveyed=TRUE (the default), telling the function to exclude such individuals from the adjacency matrix. This is done by setting variables in netvars equal to NA when the nominated id can't be found in idvar.

If the network nomination process was done in different groups (location for example) the survey id numbers may be define uniquely within each group but not across groups (there may be many individuals with id=1, for example). To encompass this issue, the user can tell the function what variable can be used to distinguish between groups through the groupvar argument. When groupvar is provided, function redifines idvar and the variables in netvars as follows:
\[
\text{dat}[[\text{idvar}]] \leftarrow \text{dat}[[\text{idvar}]] + \text{dat}[[\text{groupvar}]] * z
\]

Where \( z = 10^\text{nchar}(\text{max} (\text{dat}[[\text{idvar}]])) \).

For longitudinal data, it is assumed that the \( \text{tovar} \) holds the same information through time, this is, time-invariable. This as the package does not yet support variable times of adoption.

The \text{fill.missing} option can take any of these three values: "edgelist", "dat", or "both". This argument works as follows:

1. When \text{fill.missing="edgelist"} (or "both") the function will check which vertices show in \text{dat} but do not show in \text{edgelist}. If there is any, the function will include these in \text{edgelist} as ego to \text{NA} (so they have no link to anyone), and, if specified, will fill the \( t0, t1 \) vectors with \text{NAs} for those cases. If \( w \) is also specified, the new vertices will be set to \text{min}(\text{w}, \text{na.rm=TRUE}).

2. When \text{fill.missing="dat"} (or "both") the function checks which vertices show in \text{edgelist} but not in \text{dat}. If there is any, the function will include these in \text{dat} by adding one row per individual.

\textbf{Value}

A \texttt{diffnet} object.

\textbf{Author(s)}

Vega Yon

\textbf{See Also}

\texttt{fakesurvey, fakesurveyDyn}

Other data management functions: \texttt{diffnet-class, edgelist_to_adjmat, egonetAttrs, isolated}

\textbf{Examples}

```r
# Loading a fake survey (data frame)
data(fakesurvey)

# Diffnet object keeping isolated vertices
dn1 <- survey_to_diffnet(fakesurvey, "id", c("net1", "net2", "net3"), "toa", "group", keep.isolates=TRUE)

# Diffnet object NOT keeping isolated vertices
dn2 <- survey_to_diffnet(fakesurvey, "id", c("net1", "net2", "net3"), "toa", "group", keep.isolates=FALSE)

# dn1 has an extra vertex than dn2
dn1
dn2

# Loading a longitudinal survey data (two waves)
data(fakesurveyDyn)

groupvar <- "group"
```
x <- survey_to_diffnet(
    fakesurveyDyn, "id", c("net1", "net2", "net3"), "toa", "group",
    timevar = "time", keep.isolates = TRUE, warn.coercion=FALSE)

plot_diffnet(x, vertex.label = rownames(x))

# Reproducing medInnovationsDiffNet object ----------------------------
data(medInnovations)

# What are the netvars
netvars <- names(medInnovations)[grepl("\^net", names(medInnovations))]

medInnovationsDiffNet2 <- survey_to_diffnet(
    medInnovations,
    "id", netvars, "toa", "city",
    warn.coercion=FALSE)

medInnovationsDiffNet2

# Comparing with the package's version
all(diffnet.toa(medInnovationsDiffNet2) == diffnet.toa(medInnovationsDiffNet)) #TRUE
all(
    diffnet.attrs(medInnovationsDiffNet2, as.df = TRUE) ==
    diffnet.attrs(medInnovationsDiffNet, as.df = TRUE),
    na.rm=TRUE) #TRUE

---

threshold

Retrieve threshold levels from the exposure matrix

**Description**

Thresholds are each vertexes exposure at the time of adoption. Substantively it is the proportion of adopters required for each ego to adopt. (see exposure).

**Usage**

```r
threshold(obj, toa, t0 = min(toa, na.rm = TRUE),
    include_censored = FALSE, lags = 0L, ...)
```

**Arguments**

- `obj`: Either a $n \times T$ matrix (exposure to the innovation obtained from exposure) or a diffnet object.
- `toa`: Integer vector. Indicating the time of adoption of the innovation.
- `t0`: Integer scalar. See toa_mat.
- `include_censored`: Logical scalar. When TRUE (default), threshold
threshold

lags Integer scalar. Number of lags to consider when computing thresholds. lags=1 defines threshold as exposure at $T - 1$, where $T$ is time of adoption. levels are not reported for observations adopting in the first time period.

Further arguments to be passed to exposure.

Details

By default exposure is not computed for vertices adopting at the first time period, include_censored=FALSE, as estimating threshold for left censored data may yield biased outcomes.

Value

A vector of size $n$ indicating the threshold for each node.

Author(s)

George G. Vega Yon & Thomas W. Valente

See Also

Threshold can be visualized using plot_threshold

Other statistics: bass, classify_adopters, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, infection, moran, struct_equiv, vertex_covariate_dist

Examples

# Generating a random graph with random Times of Adoption
set.seed(783)
toa <- sample.int(4, 5, TRUE)
graph <- rgraph_er(n=5, t=max(toa) - min(toa) + 1)

# Computing exposure using Structural Equivalent
adopt <- toa_mat(toa)
se <- struct_equiv(graph)
se <- lapply(se, function(x) methods::as((x$SE)^(-1), "dgCMatrix"))
expo <- exposure(graph, adopt$cumadopt, alt.graph=se)

# Retrieving threshold
threshold(expo, toa)

# We can do the same by creating a diffnet object
diffnet <- as_diffnet(graph, toa)
threshold(diffnet, alt.graph=se)
**toa_diff**

*Difference in Time of Adoption (TOA) between individuals*

**Description**

Creates $n \times n$ matrix indicating the difference in times of adoption between each pair of nodes.

**Usage**

```r
toa_diff(obj, t0 = NULL, labels = NULL)
```

**Arguments**

- `obj`: Either an integer vector of size $n$ containing time of adoption of the innovation, or a `diffnet` object.
- `t0`: Integer scalar. Sets the lower bound of the time window (e.g. 1955).
- `labels`: Character vector of size $n$. Labels (ids) of the vertices.

**Details**

Each cell $ij$ of the resulting matrix is calculated as $toa_j - toa_i$, so that whenever its positive it means that the $j$-th individual (alter) adopted the innovation sooner.

**Value**

An $n \times n$ symmetric matrix indicating the difference in times of adoption between each pair of nodes.

**Author(s)**

George G. Vega Yon & Thomas W. Valente

**Examples**

```r
# Generating a random vector of time
set.seed(123)
times <- sample(2000:2005, 10, TRUE)

# Computing the TOA differences
toa_diff(times)
```
Time of adoption matrix

Description

Creates two matrices recording times of adoption of the innovation. One matrix records the time period of adoption for each node with zeros elsewhere. The second records the cumulative time of adoption such that there are ones for the time of adoption and every time period thereafter.

Usage

toa_mat(obj, labels = NULL, t0 = NULL, t1 = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Either an integer vector of size ( n ) containing time of adoption of the innovation, or a <code>diffnet</code> object.</td>
</tr>
<tr>
<td>labels</td>
<td>Character vector of size ( n ). Labels (ids) of the vertices.</td>
</tr>
<tr>
<td>t0</td>
<td>Integer scalar. Sets the lower bound of the time window (e.g. 1955).</td>
</tr>
<tr>
<td>t1</td>
<td>Integer scalar. Sets the upper bound of the time window (e.g. 2000).</td>
</tr>
</tbody>
</table>

Details

In order to be able to work with time ranges other than \( 1, \ldots, T \) the function receives as input the boundary labels of the time windows through the variables \( t0 \) and \( t1 \). While by default the function assumes that the the boundaries are given by the range of the `times` vector, the user can set a personalized time range exceeding the one given by the `times` vector. For instance, times of adoption may range between 2001 and 2005 but the actual data, the network, is observed between 2000 and 2005 (so there is not left censoring in the data), hence, the user could write:

```r
adopmats <- toa_mat(times, t0=2000, t1=2005)
```

That way the resulting `cumadopt` and `adopt` matrices would have \( 2005 - 2000 + 1 = 6 \) columns instead of \( 2005 - 2001 + 1 = 5 \) columns, with the first column of the two matrices containing only zeros (as the first adoption happen after the year 2000).

Value

A list of two \( n \times T \)

- `cumadopt` has 1’s for all years in which a node indicates having the innovation.
- `adopt` has 1’s only for the year of adoption and 0 for the rest.

Author(s)

George G. Vega Yon & Thomas W. Valente
Examples

# Random set of times of adoptions
times <- sample(c(NA, 2001:2005), 10, TRUE)
toa_mat(times)

# Now, suppose that we observe the graph from 2000 to 2006
toa_mat(times, t0=2000, t1=2006)

transformGraphBy  Apply a function to a graph considering non-diagonal structural zeros

Description

When there are structural zeros given by groups, this function applies a particular transformation function of a graph by groups returning a square matrix of the same size of the original one with structural zeros and the function applied by INDICES.

Usage

transformGraphBy(graph, INDICES, fun = function(g, ...) g, ...)

## S3 method for class 'diffnet'
transformGraphBy(graph, INDICES, fun = function(g, ...) g, ...)

## S3 method for class 'dgCMatrix'
transformGraphBy(graph, INDICES, fun = function(g, ...) g, ...)

Arguments

- **graph**: A graph
- **INDICES**: A vector of length n.
- **fun**: A function. This function must return a matrix of class dgCMatrix with the same dimension as dim(g).
- **...**: Further arguments passed to fun

Details

The transformation function fun must return a square matrix of size $m \times m$, where $m$ is the size of the subgroup given by INDICES. See examples below.
Examples

# Rewiring a graph by community

# Two Random graphs of different size
set.seed(123)
g0 <- rgraph_ba(m=2, self=FALSE)
g1 <- rgraph_ba(m=3, t=19, self=FALSE)

# Need a place to store both networks together!
G <- methods::new(
  Class = "dgCMatrix",
  Dim = c(1L, 1L)*((nnodes(g0) + nnodes(g1)),
  p = rep(0L, (nnodes(g0) + nnodes(g1)) + 1L)
)

# Filling the matrix
G[1:nodes(g0), 1:nodes(g0)] <- g0
G[(nnodes(g0) + 1):nnodes(G), (nnodes(g0) + 1):nnodes(G)] <- g1

# Creating an index (community)
indx <- c(rep(1, nnodes(g0)), rep(2, nnodes(g1)))

# Apply the rewiring algorithm per group
ans <- transformGraphBy(G, indx, function(g, ...) {
  rewire_graph(g, 100, "swap")
})

ans

vertex_covariate_compare

Comparisons at dyadic level

Description

Comparisons at dyadic level

Usage

vertex_covariate_compare(graph, X, funname)

Arguments

graph A matrix of size $n \times n$ of class dgCMatrix.
X A numeric vector of length $n$.
funname Character scalar. Comparison to make (see details).
This auxiliary function takes advantage of the sparseness of graph and applies a function in the form of \( \text{funname}(x_i, x_j) \) only to \((i, j)\) that have no empty entry. In other words, applies a compares elements of X only between vertices that have a link; making \( \text{nlinks(graph)} \) comparisons instead of looping through \( n \times n \), which is much faster.

funname can take any of the following values: "distance", "^2" or "quaddistance", ">" or "greater", "<" or "smaller", "=" or "equal", "^=" or "greater_equal", "\"" or "smaller_equal", "\"]" or "equal".

Value

A matrix \( \text{dgCM}\text{Matrix} \) of size \( n \times n \) with values in the form of \( \text{funname}(x_i, x_j) \).

See Also

Other dyadic-level comparison functions: \text{matrix_compare}, \text{vertex_covariate_dist}

Examples

```r
# Basic example -----------------------------------------------
set.seed(1313)
G <- rgraph_ws(10, 4, .2)
x <- rnorm(10)

vertex_covariate_compare(G, x, "distance")
vertex_covariate_compare(G, x, "^2")
vertex_covariate_compare(G, x, ">=")
vertex_covariate_compare(G, x, "<=")
```

vertex_covariate_dist  Computes covariate distance between connected vertices

Description

Computes covariate distance between connected vertices

Usage

\[
\text{vertex\_covariate\_dist}(\text{graph}, \ X, \ p = 2)
\]

\[
\text{vertex\_mahalanobis\_dist}(\text{graph}, \ X, \ S)
\]

Arguments

- **graph** A square matrix of size \( n \) of class \text{dgCM}\text{Matrix}.
- **X** A numeric matrix of size \( n \times K \). Vertices attributes
- **p** Numeric scalar. Norm to compute
- **S** Square matrix of size \( \text{ncol}(x) \). Usually the var-covar matrix.
Details

Faster than `dist`, these functions compute distance metrics between pairs of vertices that are connected (otherwise skip).

The function `vertex_covariate_dist` is the simil of `dist` and returns p-norms (Minkowski distance). It is implemented as follows (for each pair of vertices):

\[
D_{ij} = \left( \sum_{k=1}^{K} |X_{ik} - X_{jk}|^p \right)^{1/p} \quad \text{if } \text{graph}_{i,j} \neq 0
\]

In the case of mahalanobis distance, for each pair of vertex \((i,j)\), the distance is computed as follows:

\[
D_{ij} = \left( (X_i - X_j) \times S \times (X_i - X_j)' \right)^{1/2} \quad \text{if } \text{graph}_{i,j} \neq 0
\]

Value

A matrix of size \(n \times n\) of class `dgCMatrix`. Will be symmetric only if `graph` is symmetric.

Author(s)

George G. Vega Yon

References


See Also

`mahalanobis` in the stats package.

Other statistics: `bass, classify_adopters, cumulative_adopt_count, dgr, ego_variance, exposure, hazard_rate, infection, moran, struct_equiv, threshold`

Other dyadic-level comparison functions: `matrix_compare, vertex_covariate_compare`

Examples

```r
# Distance (aka p norm) -----------------------------------------------------------
set.seed(123)
G <- rgraph_ws(20, 4, .1)
X <- matrix(rnorm(40), ncol=2)
vertex_covariate_dist(G, X)

# Mahalanobis distance -------------------------------------------------------------
S <- var(X)
M <- vertex_mahalanobis_dist(G, X, S)
```
weighted_var

# Example with diffnet objects ---------------------------------------------

data(medInnovationsDiffNet)
X <- cbind(
    medInnovationsDiffNet["proage"],
    medInnovationsDiffNet["attend"]
)

S <- var(X, na.rm=TRUE)
ans <- vertex_mahalanobis_dist(medInnovationsDiffNet, X, S)

-------------------------------------------------

**weighted_var**

*Computes weighted variance*

**Description**

Computes weighted variance

**Usage**

```r
weighted_var(x, w)
wvar(x, w)
```

**Arguments**

- `x` A numeric vector of length `n`.
- `w` A numeric vector of length `n`.

**Details**

`weighted_variance` implements weighted variance computation in the following form:

\[
\frac{\sum_i w'_i (x_i - \bar{x})^2}{(1 - n)}
\]

where \(w'_i = \frac{w_i}{\sum_i w_i}\), and \(\bar{x} = \sum_i w'_i x_i\).

**Value**

Numeric scalar with the weighted variance.

**See Also**

This function is used in `diffmap`. 
Matrix multiplication methods, including diffnet objects. This function creates a generic method for %**% allowing for multiplying diffnet objects.

Usage

\[
x \ %**% \ y
\]

## Default S3 method:
\[
x \ %**% \ y
\]

## S3 method for class 'diffnet'
\[
x \ %**% \ y
\]

Arguments

- \(x\) Numeric or complex matrices or vectors, or diffnet objects.
- \(y\) Numeric or complex matrices or vectors, or diffnet objects.

Details

This function can be useful to generate alternative graphs, for example, users could compute the \(n\)-steps graph by doing \(\text{net} \ %**% \ \text{net}\) (see examples).

Value

In the case of diffnet objects performs matrix multiplication via mapply using \(x\)$graph and \(y\)$graph as arguments, returnling a diffnet. Otherwise returns the default according to %**%.

See Also

Other diffnet methods: \texttt{as.array.diffnet, c.diffnet, diffnet-arithmetic, diffnet-class, diffnet_index, plot.diffnet, summary.diffnet}

Examples

```r
# Finding the Simmelian Ties network
# Random diffnet graph
set.seed(773)
net <- rdiffnet(100, 4, seed.graph='small-world', rgraph.args=list(k=8))
netsim <- net

# According to Dekker (2006), Simmelian ties can be computed as follows
```
netsim <- net * t(net) # Keeping mutual
netsim <- netsim %*% netsim

# Checking out differences (netsim should have less)
nlinks(net)
nlinks(netsim)

mapply(`-`, nlinks(net), nlinks(netsim))
Index

+Topic datasets
  brfarmers, 12
+Topic distribution
  rgraph_ba, 123
  rgraph_er, 125
+Topic dpplot
  edges_coords, 47
  grid_distribution, 60
+Topic hplot
  plot_diffnet, 101
  plot_infectsuscep, 105
  plot_threshold, 107
+Topic manip
  edgelist_to_adjmat, 44
  isolated, 66
  toa_diff, 143
  toa_mat, 144
+Topic misc
  drawColorKey, 43
  edges_coords, 47
  grid_distribution, 60
  pretty_within, 110
  recode, 116
+Topic univar
  cumulative_adopt_count, 23
  dgr, 24
  exposure, 53
  hazard_rate, 61
  infection, 64
  struct_equiv, 131
  threshold, 141
*.diffnet (diffnet-arithmetic), 27
-.diffnet (diffnet-arithmetic), 27
/.diffnet (diffnet-arithmetic), 27
[.diffnet (diffnet_index), 34
[<.diffnet (diffnet_index), 34
[[.diffnet (diffnet_index), 34
[[<.diffnet (diffnet_index), 34
&.diffnet (diffnet-arithmetic), 27

%*, 6, 18, 28, 32, 36, 99, 138, 150
^*.diffnet (diffnet-arithmetic), 27

adjmat_to_edgelist
  (edgelist_to_adjmat), 44
adopt_changes (select_egoalter), 130
ape::Moran.I, 88
approx_geodesic, 4, 86, 132, 137
approx_geodist (approx_geodesic), 4
array, 91
arrows, 48, 109
as.array.diffnet, 5, 18, 28, 32, 36, 99, 138, 150
as.data.frame, 20
as.data.frame.diffnet, 37
as.data.frame.diffnet (diffnet-class), 28
as.data.frame.diffnet_adopters
  (classify_adopters), 19
as.dgCMatrix (as_dgCMatrix), 6
as.factor, 45, 139
as.matrix.network, 7
as_adj, 7
as_dgCMatrix, 6
as_diffnet, 22, 63, 111
as_diffnet (diffnet-class), 28
as_spmat (as_dgCMatrix), 6
bass, 8, 21, 23, 25, 52, 55, 63, 66, 89, 133, 142, 148
bass_df (bass), 8
bass_F (bass), 8
bass_f (bass), 8
bernoulli (rgraph_er), 125
binary-functions (matrix_compare), 81
boot, 10, 135
bootnet, 10, 89, 136
brfarmers, 12, 17, 40, 57–60, 80, 85
brfarmersDiffNet, 17, 17, 40, 57–60, 80, 85
INDEX

c.diffnet, 6, 18, 28, 32, 36, 99, 138, 150
c.diffnet_bootnet (bootnet), 10
c.diffnet_struct_test (struct_test), 133
classify (classify_adopters), 19
classify_adopters, 9, 19, 23, 25, 52, 55, 63, 66, 89, 133, 142, 148
classify_graph, 22
colnames, 91
colorRamp, 104
compare_matrix (matrix_compare), 81
complete_cases, 45
CUG (permute_graph), 97
cumulative_adopt_count, 9, 21, 23, 25, 52, 55, 63, 66, 86, 89, 101, 103, 105, 107, 109, 118, 133, 142, 148
diag_expand, 25
diffmap, 105, 129, 149
diffmap (diffusionMap), 40
diffnet, 6, 17, 18, 23, 37, 61, 63, 80, 85, 91, 94, 99, 113, 134, 137, 140, 143, 144, 150
diffnet (diffnet-class), 28
diffnet-arithmetic, 27
diffnet-class, 28
diffnet.attrs (diffnet-class), 28
diffnet.attrs <- (diffnet-class), 28
diffnet.toa (diffnet-class), 28
diffnet.toa <- (diffnet-class), 28
diffnet_check_attr.class, 34
diffnet_index, 6, 18, 28, 32, 34, 34, 99, 138, 150
diffnet_to_igraph (igraph), 63
diffnet_to_network (network), 94
diffnet_to_networkDynamic (network), 94
diffnetLapply (diffnet-class), 28
diffreg, 37
diffusion-data, 38
diffusionMap, 25, 40, 44, 61, 63, 101, 103–105, 107, 109, 118
dim.diffnet (diffnet-class), 28
dimnames, 91
dimnames.diffnet (diffnet-class), 28
dist, 148
distances, 137
drawColorKey, 25, 41, 43, 61, 63, 101, 103–105, 107, 109, 118
drop_isolated (isolated), 66
edgelist_to_adjmat, 32, 44, 51, 67, 117, 138, 140
edgelist_to_diffnet
  (survey_to_diffnet), 138
ditudes_coords, 47
dgo_variance, 9, 21, 23, 25, 52, 55, 63, 66, 89, 133, 142, 148
donet_attrs, 32, 46, 49, 67, 140
exposure, 9, 21, 23, 25, 52, 53, 63, 66, 89, 108, 109, 111, 133, 141, 142, 148
Extract, 30
fakeDynEdgelist, 17, 40, 57, 58–60, 80, 85
fakeEdgelist, 17, 40, 57, 58, 59, 60, 80, 85
fakesurvey, 17, 40, 57, 58, 60, 80, 85, 140
fakesurveyDyn, 17, 40, 57–59, 59, 80, 85, 140
filled.contour, 106
fitbass (bass), 8
table, 20
table.diffnet_adopters
  (classify_adopters), 19
Geodesic (approx_geodesic), 4
godist, 137
getOption, 91
glm, 37, 38
graph_power (diffnet-arithmetic), 27
grid_distribution, 25, 41, 44, 60, 63, 101, 103, 105–107, 109, 118
hazard_rate, 9, 21, 23, 25, 41, 44, 52, 55, 61, 66, 89, 101, 103, 105, 107, 109, 118, 133, 142, 148
hist, 11, 24, 134, 135
hist.default, 135
hist.diffnet_bootnet (bootnet), 10
hist.diffnet_struct_test (struct_test), 133
histogram, 24
igraph, 63, 63, 95, 115, 116, 118
igraph_to_diffnet (igraph), 63
igraph_vertex_rescale
  (rescale_vertex_igraph), 117
INDEX

print.diffnet_struct_test  
(struct_test), 133
QAP (permute_graph), 97
rank, 86, 87
rdiffnet, 54, 98, 111, 121, 122, 124, 126, 128, 129
rdiffnet_multiple (rdiffnet), 111
read_dl (read_pajek), 114
read_ml (read_pajek), 114
read_net (read_pajek), 114
read_pajek, 64, 95, 114, 116
read_ucinet (read_ucinet_head), 115
read_ucinet_head, 64, 95, 115, 115
recode, 45, 116
rect, 43
resample_graph, 11
resample_graph (bootnet), 10
rescale_vertex_igraph, 25, 41, 44, 61, 63, 86, 87, 99, 101–105, 107, 109, 117
rewire_graph, 98, 111–113, 119, 124, 126, 128, 129, 134, 135
rewire_permute (permute_graph), 97
rewire_qap (permute_graph), 97
rgraph_ba, 98, 112, 113, 122, 123, 126, 128, 129
rgraph_er, 98, 112, 113, 122, 124, 125, 128, 129
rgraph_ws, 98, 112, 113, 122, 124, 126, 127, 128, 129
ring_lattice, 98, 113, 122, 124, 126–128, 128
round, 20
round_to_seq, 41, 129
rownames, 45
rownames, 91
sapply, 111, 113
scale-free (rgraph_ba), 123
select_egoalter, 130
Shortest-Path (approx_geodesic), 4
small-world (rgraph_ws), 127
sna:::geodist, 4
sprintf, 102
str.diffnet (diffnet-class), 28
structequiv, 9, 21, 23, 25, 52–55, 63, 66, 89, 131, 142, 148
struct.test, 12, 52, 89, 120, 121, 133
struct_test_asymp (struct_test), 133
summary.diffnet, 6, 18, 28, 32, 36, 99, 137, 150
summary.diffnet adoptingChanges
(select_egoalter), 130
survey_to_diffnet, 32, 46, 51, 58, 59, 67, 138
susceptibility, 107
susceptibility (infection), 64
symbols, 48, 118
t.diffnet (diffnet-class), 28
table, 20
text, 43, 108
threshold, 9, 20, 21, 23, 25, 52, 55, 63, 66, 89, 108, 109, 133, 141, 148
title, 86
toa_diff, 143
toa_mat, 20, 23, 30, 31, 53, 61, 64, 106, 130, 141, 144
transformGraphBy, 145
UCINET (read_ucinet_head), 115
ucinet (read_ucinet_head), 115
vertex_covariate_compare, 52, 82, 146, 148
vertex_covariate_dist, 9, 21, 23, 25, 52, 55, 63, 66, 82, 89, 133, 142, 147, 147
vertex_mahalanobis_dist
(vertext_covariate_dist), 147
vertex_rescale_igraph
(rescale_vertex_igraph), 117
weighted_var, 149
wvar (weighted_var), 149