Package 'network'

December 5, 2023

Version 1.18.2

Date 2023-12-04

Title Classes for Relational Data

Maintainer Carter T. Butts <buttsc@uci.edu>

Depends R (>= 2.10), utils

Imports tibble, magrittr, statnet.common (>= 4.5), stats

Suggests sna, testthat, covr

Description Tools to create and modify network objects. The network class can represent a range of relational data types, and supports arbitrary vertex/edge/graph attributes.

License GPL (>= 2)

URL https://statnet.org/

RoxygenNote 7.2.0

Collate 'access.R' 'as.edgelist.R' 'assignment.R' 'coercion.R' 'constructors.R' 'dataframe.R' 'fileio.R' 'layout.R' 'misc.R' 'network-package.R' 'operators.R' 'plot.R' 'printsum.R' 'zzz.R'

Encoding UTF-8

NeedsCompilation yes

Author Carter T. Butts [aut, cre],
David Hunter [ctb],
Mark Handcock [ctb],
Skye Bender-deMoll [ctb],
Jeffrey Horner [ctb],
Li Wang [ctb],
Pavel N. Krivitsky [ctb] (<https://orcid.org/0000-0002-9101-3362>),
Brendan Knapp [ctb] (<https://orcid.org/0000-0003-3284-4972>),
Michał Bojanowski [ctb] (<https://orcid.org/0000-0001-7503-852X>),
Chad Klumb [ctb]

Repository CRAN

Date/Publication 2023-12-05 11:30:02 UTC

R topics documented:

network-package *Classes for Relational Data*

Description

Tools to create and modify network objects. The network class can represent a range of relational data types, and supports arbitrary vertex/edge/graph attributes.

Details

The network package provides tools for creation, access, and modification of network class objects. These objects allow for the representation of more complex structures than can be readily handled by other means (e.g., adjacency matrices), and are substantially more efficient in handling large, sparse networks. While the full capabilities of the network class can only be exploited by means of the various custom interface methods (see below), many simple tasks are streamlined through the use of operator overloading; in particular, network objects can often be treated as if they were adjacency matrices (a representation which will be familiar to users of the sna package). network objects are compatible with the sna package, and are required for many packages in the statnet bundle.

Basic information on the creation of network objects can be found by typing help(network). To learn about setting, modifying, or deleting network, vertex, or edge attributes, see help(attribute.methods). For information on custom network operators, type help(network.operators); information on overloaded operators can be found via help(network.extraction). Additional help topics are listed below.

Package:	network
Version:	1.14
Date:	May 7, 2016
Depends:	R (>= 2.10), utils
Suggests:	sna, statnet.common (>= 3.1-0)
License:	GPL (>=2)

Index of documentation pages:

flo	Florentine Wedding Data (Padgett)
get.edges	Retrieve Edges or Edge IDs Associated with a Given Vertex
get.inducedSubgraph	Retrieve Induced Subgraphs and Cuts
get.neighborhood	Obtain the Neighborhood of a Given Vertex
is.adjacent	Determine Whether Two Vertices Are Adjacent
loading.attributes	Examples of how to load vertex and edge attributes into networks
missing.edges	Identifying and Counting Missing Edges in a Network Object
network	Network Objects
network.arrow	Add Arrows or Segments to a Plot
network.density	Compute the Density of a Network
network.dyadcount	Return the Number of (Possibly Directed) Dyads in a Network Object
network.edgecount	Return the Number of Edges in a Network Object
network.edgelabel	Plots a label corresponding to an edge in a network plot.
network.extraction	Extraction and Replacement Operators for Network Objects
network.indicators	Indicator Functions for Network Properties
network.initialize	Initialize a Network Class Object
network.layout	Vertex Layout Functions for plot.network
network.loop	Add Loops to a Plot
network.operators	Network Operators
network-package	Classes for Relational Data
network.size	Return the Size of a Network
network.vertex	Add Vertices to a Plot
permute.vertexIDs	Permute (Relabel) the Vertices Within a Network
plotArgs.network	Expand and transform attributes of networks to values appropriate for aguments to plot.network
plot.network.default	Two-Dimensional Visualization for Network Objects
prod.network	Combine Networks by Edge Value Multiplication
read.paj	Read a Pajek Project or Network File and Convert to an R 'Network' Object
sum.network	Combine Networks by Edge Value Addition
valid.eids	Get the valid edge which are valid in a network
which.matrix.type	Heuristic Determination of Matrix Types for Network Storage

Author(s)

Carter T. Butts buttsc@uci.edu, with help from Mark S. Handcock handcock@stat.ucla.edu, David Hunter dhunter@stat.psu.edu, Martina Morris morrism@u.washington.edu, Skye Bender-deMoll skyebend@u.washington.edu, and Jeffrey Horner jeffrey.horner@gmail.com.

Maintainer: Carter T. Butts buttsc@uci.edu

add.edges *Add Edges to a Network Object*

Description

Add one or more edges to an existing network object.

Usage

```
add.edge(
  x,
  tail,
  head,
  names.eval = NULL,
  vals.eval = NULL,
  edge.check = FALSE,
  ...
)

add.edges(x, tail, head, names.eval = NULL, vals.eval = NULL, ...)
```

Arguments

x	an object of class network
tail	for add.edge, a vector of vertex IDs reflecting the tail set for the edge to be added; for add.edges, a list of such vectors
head	for add.edge, a vector of vertex IDs reflecting the head set for the edge to be added; for add.edges, a list of such vectors
names.eval	for add.edge, an optional list of names for edge attributes; for add.edges, a list of length equal to the number of edges, with each element containing a list of names for the attributes of the corresponding edge
vals.eval	for add.edge, an optional list of edge attribute values (matching names.eval); for add.edges, a list of such lists
edge.check	logical; should we perform (computationally expensive) tests to check for the legality of submitted edges?
...	additional arguments

Details

The edge checking procedure is very slow, but should always be employed when debugging; without it, one cannot guarantee that the network state is consistent with network level variables (see network.indicators). For example, by default it is possible to add multiple edges to a pair of vertices.

Edges can also be added/removed via the extraction/replacement operators. See the associated man page for details.

Value

Invisibly, add.edge and add.edges return pointers to their modified arguments; both functions modify their arguments in place..

Note

add.edges and add.edge were converted to an S3 generic funtions in version 1.9, so they actually call add.edges.network and add.edge.network by default, and may call other versions depending on context (i.e. when called with a networkDynamic object).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network, add.vertices, network.extraction, delete.edges, network.edgelist

Examples

```
#Initialize a small, empty network
g<-network.initialize(3)

#Add an edge
add.edge(g,1,2)
g

#Can also add edges using the extraction/replacement operators
#note that replacement operators are much slower than add.edges()
g[,3]<-1
g[,]

#Add multiple edges with attributes to a network

# pretend we just loaded in this data.frame from a file
# Note: network.edgelist() may be simpler for this case
elData<-data.frame(
  from_id=c("1","2","3","1","3","1","2"),
  to_id=c("1", "1", "1", "2", "2", "3", "3"),
  myEdgeWeight=c(1, 2, 1, 2, 5, 3, 9.5),
  someLetters=c("B", "W", "L", "Z", "P", "Q", "E"),
  edgeCols=c("red","green","blue","orange","pink","brown","gray"),
  stringsAsFactors=FALSE
)
```

```
valueNet<-network.initialize(3,loops=TRUE)

add.edges(valueNet,elData[,1],elData[,2],
    names.eval=rep(list(list("myEdgeWeight","someLetters","edgeCols")),nrow(elData)),
    vals.eval=lapply(1:nrow(elData),function(r){as.list(elData[r,3:5])}))

list.edge.attributes(valueNet)
```

`add.vertices` *Add Vertices to an Existing Network*

Description

`add.vertices` adds a specified number of vertices to an existing network; if desired, attributes for the new vertices may be specified as well.

Usage

```
add.vertices(x, nv, vattr = NULL, last.mode = TRUE, ...)
```

Arguments

`x`	an object of class `network`
`nv`	the number of vertices to add
`vattr`	optionally, a list of attributes with one entry per new vertex
`last.mode`	logical; should the new vertices be added to the last (rather than the first) mode of a bipartite network?
`...`	possible additional arguments to add.vertices

Details

New vertices are generally appended to the end of the network (i.e., their vertex IDs begin with `network.size(x)` an count upward). The one exception to this rule is when `x` is bipartite and `last.mode==FALSE`. In this case, new vertices are added to the end of the first mode, with existing second-mode vertices being permuted upward in ID. (`x`'s `bipartite` attribute is adjusted accordingly.)

Note that the attribute format used here is based on the internal (vertex-wise) storage method, as opposed to the attribute-wise format used by `network`. Specifically, `vattr` should be a list with one entry per new vertex, the ith element of which should be a list with an element for every attribute of the ith vertex. (If the required `na` attribute is not given, it will be automatically created.)

Value

Invisibly, a pointer to the updated `network` object; `add.vertices` modifies its argument in place.

Note

add.vertices was converted to an S3 generic funtion in version 1.9, so it actually calls add.vertices.network by default and may call other versions depending on context (i.e. when called with a networkDynamic object).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network, get.vertex.attribute, set.vertex.attribute

Examples

```
#Initialize a network object
g<-network.initialize(5)
g

#Add five more vertices
add.vertices(g,5)
g

#Create two more, with attributes
vat<-replicate(2,list(is.added=TRUE,num.added=2),simplify=FALSE)
add.vertices(g,2,vattr=vat)
g%v%"is.added"       #Values are only present for the new cases
g%v%"num.added"

#Add to a bipartite network
bip <-network.initialize(5,bipartite=3)
get.network.attribute(bip,'bipartite') # how many vertices in first mode?
add.vertices(bip,3,last.mode=FALSE)
get.network.attribute(bip,'bipartite')
```

---

as.color *Transform vector of values into color specification*

Description

Convenience function to convert a vector of values into a color specification.

Usage

```
as.color(x, opacity = 1)

is.color(x)
```

Arguments

x	vector of numeric, character or factor values to be transformed
opacity	optional numeric value in the range 0.0 to 1.0 used to specify the opacity/transparency (alpha) of the colors to be returned. 0 means fully opaque, 1 means fully transparent.

Behavior of `as.color` is as follows:

- integer numeric values: unchanged, (assumed to corespond to values of R's active `palette`)
- integer real values: will be translated to into grayscale values ranging between the max and min
- factor: integer values corresponding to factor levels will be used
- character: if values are valid colors (as determined by `is.color`) they will be returned as is. Otherwise converted to factor and numeric value of factor returned.

The optional `opacity` parameter can be used to make colors partially transparent (as a shortcut for `adjustcolor`. If used, colors will be returned as hex rgb color string (i.e. `"#00FF0080"`)

The `is.color` function checks if each character element of `x` appears to be a color name by comparing it to `colors` and checking if it is an HTML-style hex color code. Note that it will return FALSE for integer values.

These functions are used for the color parameters of `plot.network`.

Value

For `as.color`, a vector integer values (corresponding to color palette values) or character color name. For `is.color`, a logical vector indicating if each element of x appears to be a color

`as.color()` returns TRUE if x is a character in a known color format.

Examples

```
as.color(1:3)
as.color(c('a','b','c'))

# add some transparency
as.color(c('red','green','blue'),0.5) # gives "#FF000080", "#00FF0080", "#0000FF80"

is.color(c('red',1,'foo',NA,'#FFFFFF55'))
```

`as.data.frame.network` *Coerce a Network Object to a* `data.frame`

Description

The `as.data.frame` method coerces its input to a `data.frame` containing `x`'s edges or vertices.

Usage

```
## S3 method for class 'network'
as.data.frame(
  x,
  ...,
  unit = c("edges", "vertices"),
  na.rm = TRUE,
  attrs_to_ignore = "na",
  name_vertices = TRUE,
  sort_attrs = FALSE,
  store_eid = FALSE
)
```

Arguments

`x`	an object of class `network`
`...`	additional arguments
`unit`	whether a `data.frame` of edge or vertex attributes should be returned.
`na.rm`	logical; ignore missing edges/vertices when constructing the data frame?
`attrs_to_ignore`	character; a vector of attribute names to exclude from the returned `data.frame` (Default: `"na"`)
`name_vertices`	logical; for `unit="edges"`, should the `.tail` and the `.head` columns contain vertex names as opposed to vertex indices?
`sort_attrs`	logical; should the attribute columns in the returned data frame be sorted alphabetically?
`store_eid`	logical; for `unit="edges"`, should the edge ID in the network's internal representation be stored in a column `.eid`?

`as.edgelist` *Convert a network object into a numeric edgelist matrix*

Description

Constructs an edgelist in a sorted format with defined attributes.

Usage

```
## S3 method for class 'network'
as.edgelist(
  x,
  attrname = NULL,
  as.sna.edgelist = FALSE,
  output = c("matrix", "tibble"),
  ...
)

## S3 method for class 'matrix'
as.edgelist(
  x,
  n,
  directed = TRUE,
  bipartite = FALSE,
  loops = FALSE,
  vnames = seq_len(n),
  ...
)

## S3 method for class 'tbl_df'
as.edgelist(
  x,
  n,
  directed = TRUE,
  bipartite = FALSE,
  loops = FALSE,
  vnames = seq_len(n),
  ...
)

is.edgelist(x)
```

Arguments

`x`	a `network` object with additional class added indicating how it should be dispatched.

attrname	optionally, the name of an edge attribute to use for edge values; may be a vector of names if output="tibble"
as.sna.edgelist	logical; should the edgelist be returned in edgelist form expected by the sna package? Ignored if output="tibble"
output	return type: a matrix or a tibble; see as.matrix.network for the difference.
...	additional arguments to other methods
n	integer number of vertices in network, value passed to the 'n' flag on edgelist returned
directed	logical; is network directed, value passed to the 'directed' flag on edgelist returned
bipartite	logical or integer; is network bipartite, value passed to the 'bipartite' flag on edgelist returned
loops	logical; are self-loops allowed in network?, value passed to the 'loops' flag on edgelist returned
vnames	vertex names (defaults to vertex ids) to be attached to edgelist for sna package compatibility

Details

Constructs a edgelist matrix or tibble from a network, sorted tails-major order, with tails first, and, for undirected networks, tail < head. This format is required by some reverse-depending packages (e.g. ergm)

The as.matrix.network.edgelist provides similar functionality but it does not enforce ordering or set the edgelist class and so should be slightly faster.

is.edgelist tests if an object has the class 'edgelist'

Value

A matrix in which the first two columns are integers giving the tail (source) and head (target) vertex ids of each edge. The matrix will be given the class edgelist.

The edgelist has additional attributes attached to it:

- attr(,"n") the number of vertices in the original network
- attr(,"vnames") the names of vertices in the original network
- attr(,"directed") logical, was the original network directed
- attr(,"bipartite") was the original network bipartite
- attr(,"loops") does the original network contain self-loops

Note that if the attrname attribute is used the resulting edgelist matrix will have three columns. And if attrname refers to a character attribute, the resulting edgelist matrix will be character rather than numeric unless output="tibble".

Note

NOTE: this function was moved to network from the ergm package in network version 1.13

See Also

See also as.matrix.network.edgelist

Examples

```
data(emon)
as.edgelist(emon[[1]])
as.edgelist(emon[[1]],output="tibble")
# contrast with unsorted columns of
as.matrix.network.edgelist(emon[[1]])
```

---

as.matrix.network *Coerce a Network Object to Matrix or Table Form*

Description

The as.matrix methods attempt to coerce their input to a matrix in adjacency, incidence, or edgelist form. Edge values (from a stored attribute) may be used if present. as_tibble coerces into an edgelist in tibble (a type of data.frame) form; this can be especially useful if extrecting a character-type edge attribute.

Usage

```
## S3 method for class 'network'
as.matrix(x, matrix.type = NULL, attrname = NULL, ...)

## S3 method for class 'adjacency'
as.matrix.network(x, attrname=NULL,
   expand.bipartite = FALSE, ...)

## S3 method for class 'edgelist'
as.matrix.network(x, attrname=NULL,
   as.sna.edgelist = FALSE, na.rm = TRUE, ...)

## S3 method for class 'network'
as_tibble(
  x,
  attrnames = (match.arg(unit) == "vertices"),
  na.rm = TRUE,
  ...,
  unit = c("edges", "vertices"),
  store.eid = FALSE
)

as.tibble.network(
```

```
  x,
  attrnames = (match.arg(unit) == "vertices"),
  na.rm = TRUE,
  ...,
  unit = c("edges", "vertices"),
  store.eid = FALSE
)

## S3 method for class 'incidence'
as.matrix.network(x, attrname=NULL, ...)
```

Arguments

x	an object of class network
matrix.type	one of "adjacency", "incidence", "edgelist", or NULL
attrname	optionally, the name of an edge attribute to use for edge values
...	additional arguments.
expand.bipartite	logical; if x is bipartite, should we return the full adjacency matrix (rather than the abbreviated, two-mode form)?
as.sna.edgelist	logical; should the edgelist be returned in sna edglist form?
na.rm	logical; should missing edges/vertices be included in the edgelist formats? Ignored if as.sna.edgelist=TRUE.
attrnames	optionally, either a character vector of the names of edge attributes to use for edge values, or a numerical or logical vector to use as indices for selecting them from list.edge.attributes(x) or list.vertex.attributes(x) (depending on unit); passing TRUE therefore returns all edge attributes as columns
unit	whether a tibble of edge or vertex attributes should be returned.
store.eid	whether the edge ID should be stored in the third column (.eid).

Details

If no matrix type is specified, which.matrix.type will be used to make an educated guess based on the shape of x. Where edge values are not specified, a dichotomous matrix will be assumed.

Edgelists returned by the as.matrix methods are by default in a slightly different form from the sna edgelist standard, but do contain the sna extended matrix attributes (see as.network.matrix). They should typically be compatible with sna library functions. To ensure compatibility, the as.sna.edgelist argument can be set (which returns an exact sna edgelist). The as.edgelist function also returns a similar edgelist matrix but with an enforced sorting.

For the as.matrix methods, if the attrname attribute is used to include a charcter attribute, the resulting edgelist matrix will be character rather than numeric. The as_tibble methods never coerce.

Note that adjacency matrices may also be obtained using the extraction operator. See the relevant man page for details. Also note that which attributes get returned by the as_tibble method by default depends on unit: by default no edge attributes are returned but all vertex attributes are.

Value

For `as.matrix` methods, an adjacency, incidence, or edgelist matrix. For the `as_tibble` method, a `tibble` whose first two columns are `.head` and `.tail`, whose third column `.eid` is the edge ID, and whose subsequent columns are the requested edge attributes.

Author(s)

Carter T. Butts <buttsc@uci.edu> and David Hunter <dhunter@stat.psu.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`which.matrix.type`, `network`, `network.extraction`,`as.edgelist`

Examples

```
# Create a random network
m <- matrix(rbinom(25,4,0.159),5,5) # 50% density
diag(m) <- 0
g <- network(m, ignore.eval=FALSE, names.eval="a") # With values
g %e% "ac" <- letters[g %e% "a"]

# Coerce to matrix form
# No attributes:
as.matrix(g,matrix.type="adjacency")
as.matrix(g,matrix.type="incidence")
as.matrix(g,matrix.type="edgelist")
# Attributes:
as.matrix(g,matrix.type="adjacency",attrname="a")
as.matrix(g,matrix.type="incidence",attrname="a")
as.matrix(g,matrix.type="edgelist",attrname="a")
as.matrix(g,matrix.type="edgelist",attrname="ac")

# Coerce to a tibble:
library(tibble)
as_tibble(g)
as_tibble(g, attrnames=c("a","ac"))
as_tibble(g, attrnames=TRUE)
# Get vertex attributes instead:
as_tibble(g, unit = "vertices")

# Missing data handling:
g[1,2] <- NA
as.matrix(g,matrix.type="adjacency") # NA in the corresponding cell
as.matrix(g,matrix.type="edgelist", na.rm=TRUE) # (1,2) excluded
as.matrix(g,matrix.type="edgelist", na.rm=FALSE) # (1,2) included
```

```
as_tibble(g, attrnames="na", na.rm=FALSE) # Which edges are marked missing?

# Can also use the extraction operator
g[,]                          # Get entire adjacency matrix
g[1:2,3:5]                    # Obtain a submatrix
```

`as.network.matrix` *Coercion from Matrices to Network Objects*

Description

`as.network.matrix` attempts to coerce its first argument to an object of class `network`.

Usage

```
## Default S3 method:
as.network(x, ...)

## S3 method for class 'matrix'
as.network(
  x,
  matrix.type = NULL,
  directed = TRUE,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE,
  ignore.eval = TRUE,
  names.eval = NULL,
  na.rm = FALSE,
  edge.check = FALSE,
  ...
)
```

Arguments

`x`	a matrix containing an adjacency structure
`...`	additional arguments
`matrix.type`	one of `"adjacency"`, `"edgelist"`, `"incidence"`, or `NULL`
`directed`	logical; should edges be interpreted as directed?
`hyper`	logical; are hyperedges allowed?
`loops`	logical; should loops be allowed?
`multiple`	logical; are multiplex edges allowed?

bipartite	count; should the network be interpreted as bipartite? If present (i.e., non-NULL) it is the count of the number of actors in the bipartite network. In this case, the number of nodes is equal to the number of actors plus the number of events (with all actors preceding all events). The edges are then interpreted as nondirected.
ignore.eval	logical; ignore edge values?
names.eval	optionally, the name of the attribute in which edge values should be stored
na.rm	logical; ignore missing entries when constructing the network?
edge.check	logical; perform consistency checks on new edges?

Details

Depending on matrix.type, one of three edgeset constructor methods will be employed to read the input matrix (see edgeset.constructors). If matrix.type==NULL, which.matrix.type will be used to guess the appropriate matrix type.

The coercion methods will recognize and attempt to utilize the sna extended matrix attributes where feasible. These are as follows:

- "n": taken to indicate number of vertices in the network.
- "bipartite": taken to indicate the network's bipartite attribute, where present.
- "vnames": taken to contain vertex names, where present.

These attributes are generally used with edgelists, and indeed data in sna edgelist format should be transparently converted in most cases. Where the extended matrix attributes are in conflict with the actual contents of x, results are no guaranteed (but the latter will usually override the former). For an edge list, the number of nodes in a network is determined by the number of unique nodes specified. If there are isolate nodes not in the edge list, the "n" attribute needs to be set. See example below.

Value

An object of class network

Author(s)

Carter T. Butts <buttsc@uci.edu> and David Hunter <dhunter@stat.psu.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

edgeset.constructors, network, which.matrix.type

Examples

```
#Draw a random matrix
m<-matrix(rbinom(25,1,0.5),5)
diag(m)<-0

#Coerce to network form
g<-as.network.matrix(m,matrix.type="adjacency")

# edge list example. Only 4 nodes in the edge list.
m = matrix(c(1,2, 2,3, 3,4), byrow = TRUE, nrow=3)
attr(m, 'n') = 7
as.network(m, matrix.type='edgelist')
```

`as.sociomatrix` *Coerce One or More Networks to Sociomatrix Form*

Description

`as.sociomatrix` takes adjacency matrices, adjacency arrays, `network` objects, or lists thereof, and returns one or more sociomatrices (adjacency matrices) as appropriate. This routine provides a useful input-agnostic front-end to functions which process adjacency matrices.

Usage

```
as.sociomatrix(
  x,
  attrname = NULL,
  simplify = TRUE,
  expand.bipartite = FALSE,
  ...
)
```

Arguments

`x`	an adjacency matrix, array, `network` object, or list thereof.
`attrname`	optionally, the name of a network attribute to use for extracting edge values (if `x` is a `network` object).
`simplify`	logical; should `as.sociomatrix` attempt to combine its inputs into an adjacency array (`TRUE`), or return them as separate list elements (`FALSE`)?
`expand.bipartite`	logical; if `x` is bipartite, should we return the full adjacency matrix (rather than the abbreviated, two-mode form)?
`...`	additional arguments for the coercion routine.

Details

as.sociomatrix provides a more general means of coercing input into adjacency matrix form than as.matrix.network. In particular, as.sociomatrix will attempt to coerce all input networks into the appropriate form, and return the resulting matrices in a regularized manner. If simplify==TRUE, as.sociomatrix attempts to return the matrices as a single adjacency array. If the input networks are of variable size, or if simplify==FALSE, the networks in question are returned as a list of matrices. In any event, a single input network is always returned as a lone matrix.

If attrname is given, the specified edge attribute is used to extract edge values from any network objects contained in x. Note that the same attribute will be used for all networks; if no attribute is specified, the standard dichotomous default will be used instead.

Value

One or more adjacency matrices. If all matrices are of the same dimension and simplify==TRUE, the matrices are joined into a single array; otherwise, the return value is a list of single adjacency matrices.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

as.matrix.network, network

Examples

```
#Generate an adjacency array
g<-array(rbinom(100,1,0.5),dim=c(4,5,5))

#Generate a network object
net<-network(matrix(rbinom(36,1,0.5),6,6))

#Coerce to adjacency matrix form using as.sociomatrix
as.sociomatrix(g,simplify=TRUE)   #Returns as-is
as.sociomatrix(g,simplify=FALSE)  #Returns as list
as.sociomatrix(net)               #Coerces to matrix
as.sociomatrix(list(net,g))       #Returns as list of matrices
```

attribute.methods *Attribute Interface Methods for the Network Class*

Description

These methods get, set, list, and delete attributes at the network, edge, and vertex level.

Usage

```
delete.edge.attribute(x, attrname, ...)

## S3 method for class 'network'
delete.edge.attribute(x, attrname, ...)

delete.network.attribute(x, attrname, ...)

## S3 method for class 'network'
delete.network.attribute(x, attrname, ...)

delete.vertex.attribute(x, attrname, ...)

## S3 method for class 'network'
delete.vertex.attribute(x, attrname, ...)

get.edge.attribute(x, ..., el)

## S3 method for class 'network'
get.edge.attribute(
  x,
  attrname,
  unlist = TRUE,
  na.omit = FALSE,
  null.na = FALSE,
  deleted.edges.omit = FALSE,
  ...,
  el
)

## S3 method for class 'list'
get.edge.attribute(
  x,
  attrname,
  unlist = TRUE,
  na.omit = FALSE,
  null.na = FALSE,
  deleted.edges.omit = FALSE,
  ...,
```

```
  el
)

get.edge.value(x, ...)

## S3 method for class 'network'
get.edge.value(
  x,
  attrname,
  unlist = TRUE,
  na.omit = FALSE,
  null.na = FALSE,
  deleted.edges.omit = FALSE,
  ...
)

## S3 method for class 'list'
get.edge.value(
  x,
  attrname,
  unlist = TRUE,
  na.omit = FALSE,
  null.na = FALSE,
  deleted.edges.omit = FALSE,
  ...
)

get.network.attribute(x, ...)

## S3 method for class 'network'
get.network.attribute(x, attrname, unlist = FALSE, ...)

get.vertex.attribute(x, ...)

## S3 method for class 'network'
get.vertex.attribute(
  x,
  attrname,
  na.omit = FALSE,
  null.na = TRUE,
  unlist = TRUE,
  ...
)

list.edge.attributes(x, ...)

## S3 method for class 'network'
list.edge.attributes(x, ...)
```

```
list.network.attributes(x, ...)

## S3 method for class 'network'
list.network.attributes(x, ...)

list.vertex.attributes(x, ...)

## S3 method for class 'network'
list.vertex.attributes(x, ...)

network.vertex.names(x)

network.vertex.names(x) <- value

set.edge.attribute(x, attrname, value, e, ...)

## S3 method for class 'network'
set.edge.attribute(x, attrname, value, e = seq_along(x$mel), ...)

set.edge.value(x, attrname, value, e, ...)

## S3 method for class 'network'
set.edge.value(x, attrname, value, e = seq_along(x$mel), ...)

set.network.attribute(x, attrname, value, ...)

## S3 method for class 'network'
set.network.attribute(x, attrname, value, ...)

set.vertex.attribute(x, attrname, value, v = seq_len(network.size(x)), ...)

## S3 method for class 'network'
set.vertex.attribute(x, attrname, value, v = seq_len(network.size(x)), ...)
```

Arguments

x	an object of class network, or a list of edges (possibly network$mel) in get.edge.attribute.
attrname	the name of the attribute to get or set.
...	additional arguments
el	Deprecated; use x instead.
unlist	logical; should retrieved attribute values be unlisted prior to being returned?
na.omit	logical; should retrieved attribute values corresponding to vertices/edges marked as 'missing' be removed?
null.na	logical; should NULL values (corresponding to vertices or edges with no values set for the attribute) be replaced with NAs in output?

`deleted.edges.omit`
: logical: should the elements corresponding to deleted edges be removed?

`value`
: values of the attribute to be set; these should be in `vector` or `list` form for the `edge` and `vertex` cases, or `matrix` form for `set.edge.value`.

`e`
: IDs for the edges whose attributes are to be altered.

`v`
: IDs for the vertices whose attributes are to be altered.

Details

The `list.attributes` functions return the names of all edge, network, or vertex attributes (respectively) in the network. All attributes need not be defined for all elements; the union of all extant attributes for the respective element type is returned.

The `get.attribute` functions look for an edge, network, or vertex attribute (respectively) with the name `attrname`, returning its values. Note that, to retrieve an edge attribute from all edges within a network `x`, `x$mel` should be used as the first argument to `get.edge.attribute`; `get.edge.value` is a convenience function which does this automatically. As of v1.7.2, if a `network` object is passed to `get.edge.attribute` it will automatically call `get.edge.value` instead of returning NULL. When the parameters `na.omit`, or `deleted.edges.omit` are used, the position index of the attribute values returned will not correspond to the vertex/edge id. To preserved backward compatibility, if the edge attribute `attrname` does not exist for any edge, `get.edge.attribute` will still return `NULL` even if `null.na=TRUE`

`network.vertex.names` is a convenience function to extract the `"vertex.names"` attribute from all vertices.

The `set.attribute` functions allow one to set the values of edge, network, or vertex attributes. `set.edge.value` is a convenience function which allows edge attributes to be given in adjacency matrix form, and the assignment form of `network.vertex.names` is likewise a convenient front-end to `set.vertex.attribute` for vertex names. The `delete.attribute` functions, by contrast, remove the named attribute from the network, from all edges, or from all vertices (as appropriate). If `attrname` is a vector of attribute names, each will be removed in turn. These functions modify their arguments in place, although a pointer to the modified object is also (invisibly) returned.

Additional practical example of how to load and attach attributes are on the `loading.attributes` page.

Some attribute assignment/extraction can be performed conveniently through the various extraction/replacement operators, although they may be less efficient. See the associated man page for details.

Value

For the `list.attributes` methods, a vector containing attribute names. For the `get.attribute` methods, a list containing the values of the attribute in question (or simply the value itself, for `get.network.attribute`). For the `set.attribute` and `delete.attribute` methods, a pointer to the updated `network` object.

Note

As of version 1.9 the `set.vertex.attribute` function can accept and modify multiple attributes in a single call to improve efficiency. For this case `attrname` can be a list or vector of attribute

names and value is a list of values corresponding to the elements of attrname (can also be a list of lists of values if elements in v should have different values).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

loading.attributes,network, as.network.matrix, as.sociomatrix, as.matrix.network, network.extraction

Examples

```
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)

#Create a matrix of values corresponding to edges
mm<-m
mm[1,2]<-7; mm[2,3]<-4; mm[3,1]<-2

#Assign some attributes
set.edge.attribute(g,"myeval",3:5)
set.edge.value(g,"myeval2",mm)
set.network.attribute(g,"mygval","boo")
set.vertex.attribute(g,"myvval",letters[1:3])
network.vertex.names(g) <- LETTERS[1:10]

#List the attributes
list.edge.attributes(g)
list.network.attributes(g)
list.vertex.attributes(g)

#Retrieve the attributes
get.edge.attribute(g$mel,"myeval")  #Note the first argument!
get.edge.value(g,"myeval")          #Another way to do this
get.edge.attribute(g$mel,"myeval2")
get.network.attribute(g,"mygval")
get.vertex.attribute(g,"myvval")
network.vertex.names(g)

#Purge the attributes
delete.edge.attribute(g,"myeval")
delete.edge.attribute(g,"myeval2")
delete.network.attribute(g,"mygval")
delete.vertex.attribute(g,"myvval")
```

```
#Verify that the attributes are gone
list.edge.attributes(g)
list.network.attributes(g)
list.vertex.attributes(g)

#Note that we can do similar things using operators
g %n% "mygval" <- "boo"               #Set attributes, as above
g %v% "myvval" <- letters[1:3]
g %e% "myeval" <- mm
g[,,names.eval="myeval"] <- mm          #Another way to do this
g %n% "mygval"                        #Retrieve the attributes
g %v% "myvval"
g %e% "mevval"
as.sociomatrix(g,"myeval")              # Or like this
```

`deletion.methods` *Remove Elements from a Network Object*

Description

`delete.edges` removes one or more edges (specified by their internal ID numbers) from a network; `delete.vertices` performs the same task for vertices (removing all associated edges in the process).

Usage

```
delete.edges(x, eid, ...)

## S3 method for class 'network'
delete.edges(x, eid, ...)

delete.vertices(x, vid, ...)

## S3 method for class 'network'
delete.vertices(x, vid, ...)
```

Arguments

`x`	an object of class `network`.
`eid`	a vector of edge IDs.
`...`	additional arguments to methods.
`vid`	a vector of vertex IDs.

Details

Note that an edge's ID number corresponds to its order within `x$mel`. To determine edge IDs, see `get.edgeIDs`. Likewise, vertex ID numbers reflect the order with which vertices are listed internally (e.g., the order of `x$oel` and `x$iel`, or that used by `as.matrix.network.adjacency`). When vertices are removed from a network, all edges having those vertices as endpoints are removed as well. When edges are removed, the remaining edge ids are NOT permuted and `NULL` elements will be left on the list of edges, which may complicate some functions that require eids (such as `set.edge.attribute`). The function `valid.eids` provides a means to determine the set of valid (non-NULL) edge ids.

Edges can also be added/removed via the extraction/replacement operators. See the associated man page for details.

Value

Invisibly, a pointer to the updated network; these functions modify their arguments in place.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`get.edgeIDs`, `network.extraction`, `valid.eids`

Examples

```
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)

as.matrix.network(g)
delete.edges(g,2)              #Remove an edge
as.matrix.network(g)
delete.vertices(g,2)           #Remove a vertex
as.matrix.network(g)

#Can also remove edges using extraction/replacement operators
g<-network(m)
g[1,2]<-0                      #Remove an edge
g[,]
g[,]<-0                        #Remove all edges
g[,]
```

edgeset.constructors *Edgeset Constructors for Network Objects*

Description

These functions convert relational data in matrix form to network edge sets.

Usage

```
network.bipartite(x, g, ignore.eval = TRUE, names.eval = NULL, ...)

network.adjacency(x, g, ignore.eval = TRUE, names.eval = NULL, ...)

network.edgelist(x, g, ignore.eval = TRUE, names.eval = NULL, ...)

network.incidence(x, g, ignore.eval = TRUE, names.eval = NULL, ...)
```

Arguments

x	a matrix containing edge information
g	an object of class network
ignore.eval	logical; ignore edge value information in x?
names.eval	a name for the edge attribute under which to store edge values, if any
...	possible additional arguments (such as edge.check)

Details

Each of the above functions takes a network and a matrix as input, and modifies the supplied network object by adding the appropriate edges. network.adjacency takes x to be an adjacency matrix; network.edgelist takes x to be an edgelist matrix; and network.incidence takes x to be an incidence matrix. network.bipartite takes x to be a two-mode adjacency matrix where rows and columns reflect each respective mode (conventionally, actors and events); If ignore.eval==FALSE, (non-zero) edge values are stored as edgewise attributes with name names.eval. The edge.check argument can be added via ... and will be passed to add.edges.

Edgelist matrices to be used with network.edgelist should have one row per edge, with the first two columns indicating the sender and receiver of each edge (respectively). Edge values may be provided in additional columns. The edge attributes will be created with names corresponding to the column names unless alternate names are provided via names.eval. The vertices specified in the first two columns, which can be characters, are added to the network in default sort order. The edges are added in the order specified by the edgelist matrix.

Incidence matrices should contain one row per vertex, with one column per edge. A non-zero entry in the matrix means that the edge with the id corresponding to the column index will have an incident vertex with an id corresponding to the row index. In the directed case, negative cell values are taken to indicate tail vertices, while positive values indicate head vertices.

Results similar to network.adjacency can also be obtained by means of extraction/replacement operators. See the associated man page for details.

Value

Invisibly, an object of class network; these functions modify their argument in place.

Author(s)

Carter T. Butts <buttsc@uci.edu> and David Hunter <dhunter@stat.psu.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

loading.attributes, network, network.initialize, add.edges, network.extraction

Examples

```
#Create an arbitrary adjacency matrix
m<-matrix(rbinom(25,1,0.5),5,5)
diag(m)<-0

g<-network.initialize(5)    #Initialize the network
network.adjacency(m,g)      #Import the edge data

#Do the same thing, using replacement operators
g<-network.initialize(5)
g[,]<-m

# load edges from a data.frame via network.edgelist
edata <-data.frame(
  tails=c(1,2,3),
  heads=c(2,3,1),
  love=c('yes','no','maybe'),
  hate=c(3,-5,2),
  stringsAsFactors=FALSE
  )

g<-network.edgelist(edata,network.initialize(4),ignore.eval=FALSE)
as.sociomatrix(g,attrname='hate')
g%e%'love'

# load edges from an incidence matrix
inci<-matrix(c(1,1,0,0, 0,1,1,0, 1,0,1,0),ncol=3,byrow=FALSE)
inci
g<-network.incidence(inci,network.initialize(4,directed=FALSE))
as.matrix(g)
```

emon *Interorganizational Search and Rescue Networks (Drabek et al.)*

Description

Drabek et al. (1981) provide seven case studies of emergent multi-organizational networks (EMONs) in the context of search and rescue (SAR) activities. Networks of interaction frequency are reported, along with several organizational attributes.

Usage

```
data(emon)
```

Format

A list of 7 network objects:

[[1]]	Cheyenne	network	Cheyenne SAR EMON
[[2]]	HurrFrederic	network	Hurricane Frederic SAR EMON
[[3]]	LakePomona	network	Lake Pomona SAR EMON
[[4]]	MtSi	network	Mt. Si SAR EMON
[[5]]	MtStHelens	network	Mt. St. Helens SAR EMON
[[6]]	Texas	network	Texas Hill Country SAR EMON
[[7]]	Wichita	network	Wichita Falls SAR EMON

Each network has one edge attribute:

Frequency	numeric	Interaction frequency (1-4; 1=most frequent)

Each network also has 8 vertex attributes:

Command.Rank.Score	numeric	Mean rank in the command structure
Decision.Rank.Score	numeric	Mean rank in the decision process
Formalization	numeric	Degree of formalization
Location	character	Location code
Paid.Staff	numeric	Number of paid staff
Sponsorship	character	Sponsorship type
vertex.names	character	Organization name
Volunteer.Staff	numeric	Number of volunteer staff

Details

All networks collected by Drabek et al. reflect reported frequency of organizational interaction during the search and rescue effort; the (i,j) edge constitutes i's report regarding interaction with j, with non-adjacent vertices reporting no contact. Frequency is rated on a four-point scale, with 1 indicating the highest frequency of interaction. (Response options: 1="continuously", 2="about once an

hour", 3="every few hours", 4="about once a day or less") This is stored within the "Frequency" edge attribute.

For each network, several covariates are recorded as vertex attributes:

Command.Rank.Score Mean (reversed) rank for the prominence of each organization in the command structure of the response, as judged by organizational informants.

Decision.Rank.Score Mean (reversed) rank for the prominence of each organization in decision making processes during the response, as judged by organizational informants.

Formalization An index of organizational formalization, ranging from 0 (least formalized) to 4 (most formalized).

Localization For each organization, "L" if the organization was sited locally to the impact area, "NL" if the organization was not sited near the impact area, "B" if the organization was sited at both local and non-local locations.

Paid.Staff Number of paid staff employed by each organization at the time of the response.

Sponsorship The level at which each organization was sponsored (e.g., "City", "County", "State", "Federal", and "Private").

vertex.names The identity of each organization.

Volunteer.Staff Number of volunteer staff employed by each organization at the time of the response.

Note that where intervals were given by the original source, midpoints have been substituted. For detailed information regarding data coding and procedures, see Drabek et al. (1981).

Source

Drabek, T.E.; Tamminga, H.L.; Kilijanek, T.S.; and Adams, C.R. (1981). *Data from Managing Multiorganizational Emergency Responses: Emergent Search and Rescue Networks in Natural Disaster and Remote Area Settings.* Program on Technology, Environment, and Man Monograph 33. Institute for Behavioral Science, University of Colorado.

See Also

network

Examples

```
data(emon)   #Load the emon data set

#Plot the EMONs
par(mfrow=c(3,3))
for(i in 1:length(emon))
  plot(emon[[i]],main=names(emon)[i],edge.lwd="Frequency")
```

flo *Florentine Wedding Data (Padgett)*

Description

This is a data set of Padgett (1994), consisting of weddings among leading Florentine families. This data is stored in symmetric adjacency matrix form.

Usage

```
data(flo)
```

Source

Padgett, John F. (1994). "Marriage and Elite Structure in Renaissance Florence, 1282-1500." Paper delivered to the Social Science History Association.

References

Wasserman, S. and Faust, K. (1994) *Social Network Analysis: Methods and Applications*, Cambridge: Cambridge University Press.

See Also

network

Examples

```
data(flo)
nflo<-network(flo,directed=FALSE)    #Convert to network object form
all(nflo[,]==flo)                    #Trust, but verify
                                     #A fancy display:
plot(nflo,displaylabels=TRUE,boxed.labels=FALSE,label.cex=0.75)
```

get.edges *Retrieve Edges or Edge IDs Associated with a Given Vertex*

Description

`get.edges` retrieves a list of edges incident on a given vertex; `get.edgeIDs` returns the internal identifiers for those edges, instead. Both allow edges to be selected based on vertex neighborhood and (optionally) an additional endpoint.

Usage

```
get.edgeIDs(
  x,
  v,
  alter = NULL,
  neighborhood = c("out", "in", "combined"),
  na.omit = TRUE
)

get.edges(
  x,
  v,
  alter = NULL,
  neighborhood = c("out", "in", "combined"),
  na.omit = TRUE
)

get.dyads.eids(
  x,
  tails,
  heads,
  neighborhood = c("out", "in", "combined"),
  na.omit = TRUE
)
```

Arguments

x	an object of class network
v	a vertex ID
alter	optionally, the ID of another vertex
neighborhood	an indicator for whether we are interested in in-edges, out-edges, or both (relative to v). defaults to 'combined' for undirected networks
na.omit	logical; should we omit missing edges?
tails	a vector of vertex ID for the 'tails' (v) side of the dyad
heads	a vector of vertex ID for the 'heads' (alter) side of the dyad

Details

By default, get.edges returns all out-, in-, or out- and in-edges containing v. get.edgeIDs is identical, save in its return value, as it returns only the ids of the edges. Specifying a vertex in alter causes these edges to be further selected such that alter must also belong to the edge – this can be used to extract edges between two particular vertices. Omission of missing edges is accomplished via na.omit. Note that for multiplex networks, multiple edges or edge ids can be returned.

The function get.dyads.eids simplifies the process of looking up the edge ids associated with a set of 'dyads' (tail and head vertex ids) for edges. It only is intended for working with non-multiplex networks and will return a warning and NA value for any dyads that correspond to multiple edges. The value numeric(0) will be returned for any dyads that do not have a corresponding edge.

Value

For `get.edges`, a list of edges. For `get.edgeIDs`, a vector of edge ID numbers. For `get.dyads.eids`, a list of edge IDs corresponding to the dyads defined by the vertex ids in `tails` and `heads`

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`get.neighborhood`, `valid.eids`

Examples

```
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)

get.edges(g,1,neighborhood="out")
get.edgeIDs(g,1,neighborhood="in")
```

---

get.inducedSubgraph *Retrieve Induced Subgraphs and Cuts*

Description

Given a set of vertex IDs, `get.inducedSubgraph` returns the subgraph induced by the specified vertices (i.e., the vertices and all associated edges). Optionally, passing a second set of alters returns the cut from the first to the second set (i.e., all edges passing between the sets), along with the associated endpoints. Alternatively, passing in a vector of edge ids will induce a subgraph containing the specified edges and their incident vertices. In all cases, the result is returned as a network object, with all attributes of the selected edges and/or vertices (and any network attributes) preserved.

Usage

```
get.inducedSubgraph(x, v, alters = NULL, eid = NULL)

x %s% v
```

Arguments

x	an object of class network.
v	a vector of vertex IDs, or, for %s%, optionally a list containing two disjoint vectors of vertex IDs (see below).
alters	optionally, a second vector of vertex IDs. Must be disjoint with v.
eid	optionally, a numeric vector of valid edge ids in x that should be retained (cannot be used with v or alter)

Details

For get.inducedSubgraph, v can be a vector of vertex IDs. If alter=NULL, the subgraph induced by these vertices is returned. Calling %s% with a single vector of vertices has an identical effect.

Where alters is specified, it must be a vector of IDs disjoint with v. Where both are given, the edges spanning v and alters are returned, along with the vertices in question. (Technically, only the edges really constitute the "cut," but the vertices are included as well.) The same result can be obtained with the %s% operator by passing a two-element list on the right hand side; the first element is then interpreted as v, and the second as alters.

When eid is specified, the v and alters argument will be ignored and the subgraph induced by the specified edges and their incident vertices will be returned.

Any network, vertex, or edge attributes for the selected network elements are retained (although features such as vertex IDs and the network size will typically change). These are copies of the elements in the original network, which is not altered by this function.

Value

A network object containing the induced subgraph.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

network, network.extraction

Examples

```
#Load the Drabek et al. EMON data
data(emon)

#For the Mt. St. Helens, EMON, several types of organizations are present:
type<-emon$MtStHelens %v% "Sponsorship"

#Plot interactions among the state organizations
plot(emon$MtStHelens %s% which(type=="State"), displaylabels=TRUE)

#Plot state/federal interactions
```

```
plot(emon$MtStHelens %s% list(which(type=="State"),
    which(type=="Federal")),  displaylabels=TRUE)

#Plot state interactions with everyone else
plot(emon$MtStHelens %s% list(which(type=="State"),
    which(type!="State")), displaylabels=TRUE)

# plot only interactions with frequency of 2
subG2<-get.inducedSubgraph(emon$MtStHelens,
            eid=which(emon$MtStHelens%e%'Frequency'==2))
plot(subG2,edge.label='Frequency')
```

`get.neighborhood` *Obtain the Neighborhood of a Given Vertex*

Description

`get.neighborhood` returns the IDs of all vertices belonging to the in, out, or combined neighborhoods of `v` within network `x`.

Usage

```
get.neighborhood(x, v, type = c("out", "in", "combined"), na.omit = TRUE)
```

Arguments

`x`	an object of class `network`
`v`	a vertex ID
`type`	the neighborhood to be computed
`na.omit`	logical; should missing edges be ignored when obtaining vertex neighborhoods?

Details

Note that the combined neighborhood is the union of the in and out neighborhoods – as such, no vertex will appear twice.

Value

A vector containing the vertex IDs for the chosen neighborhood.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Wasserman, S. and Faust, K. 1994. *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

get.edges, is.adjacent

Examples

```
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)

#Examine the neighborhood of vertex 1
get.neighborhood(g,1,"out")
get.neighborhood(g,1,"in")
get.neighborhood(g,1,"combined")
```

---

has.edges *Determine if specified vertices of a network have any edges (are not isolates)*

Description

Returns a logical value for each specified vertex, indicating if it has any incident (in or out) edges. Checks all vertices by default

Usage

```
has.edges(net, v = seq_len(network.size(net)))
```

Arguments

net	a network object to be queried
v	integer vector of vertex ids to check

Value

returns a logical vector with the same length as v, with TRUE if the vertex is involved in any edges, FALSE if it is an isolate.

Author(s)

skyebend

Examples

```
test<-network.initialize(5)
test[1,2]<-1
has.edges(test)
has.edges(test,v=5)
```

---

is.adjacent *Determine Whether Two Vertices Are Adjacent*

Description

`is.adjacent` returns `TRUE` iff `vi` is adjacent to `vj` in `x`. Missing edges may be omitted or not, as per `na.omit`.

Usage

```
is.adjacent(x, vi, vj, na.omit = FALSE)
```

Arguments

`x`	an object of class `network`
`vi`	a vertex ID
`vj`	a second vertex ID
`na.omit`	logical; should missing edges be ignored when assessing adjacency?

Details

Vertex v is said to be adjacent to vertex v' within directed network G iff there exists some edge whose tail set contains v and whose head set contains v'. In the undirected case, head and tail sets are exchangeable, and thus v is adjacent to v' if there exists an edge such that v belongs to one endpoint set and v' belongs to the other. (In dyadic graphs, these sets are of cardinality 1, but this may not be the case where hyperedges are admitted.)

If an edge which would make v and v' adjacent is marked as missing (via its `na` attribute), then the behavior of `is.adjacent` depends upon `na.omit`. If `na.omit==FALSE` (the default), then the return value is considered to be `NA` unless there is also *another* edge from v to v' which is *not* missing (in which case the two are clearly adjacent). If `na.omit==TRUE`, on the other hand the missing edge is simply disregarded in assessing adjacency (i.e., it effectively treated as not present). It is important not to confuse "not present" with "missing" in this context: the former indicates that the edge in question does not belong to the network, while the latter indicates that the state of the corresponding

edge is regarded as unknown. By default, all edge states are assumed "known" unless otherwise indicated (by setting the edge's na attribute to TRUE; see attribute.methods).

Adjacency can also be determined via the extraction/replacement operators. See the associated man page for details.

Value

A logical, giving the status of the (i,j) edge

Note

Prior to version 1.4, na.omit was set to TRUE by default.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Wasserman, S. and Faust, K. 1994. *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

get.neighborhood, network.extraction, attribute.methods

Examples

```
#Create a very simple graph
g<-network.initialize(3)
add.edge(g,1,2)
is.adjacent(g,1,2)  #TRUE
is.adjacent(g,2,1)  #FALSE
g[1,2]==1           #TRUE
g[2,1]==1           #FALSE
```

---

loading.attributes *Examples of how to load vertex and edge attributes into networks*

Description

Additional examples of how to manipulate network attributes using the functions documented in attribute.methods

Details

The attribute.methods documentation gives details about the use of the specific network attribute methods such as get.vertex.attribute and set.edge.attribute. This document gives examples of how to load in and attach attribute data, drawing heavily on material from the Sunbelt statnet workshops https://statnet.org/workshops/.

The examples section below give a quick overview of:

- Loading in a matrix
- Attaching vertex attributes
- Attaching edge atributes from a matrix
- Loading in an edgelist
- Attaching edge atributes from an edgelist

The read.table documentation provides more information about reading data in from various tabular file formats prior to loading into a network. Note that the output is usually a data.frame object in which each columns is represented as a factor. This means that in some cases when the output is directly loaded into a network the variable values will appear as factor level numbers instead of text values. The stringsAsFactors=FALSE flag may help with this, but some columns may need to be converted using as.numeric or as.character where appropriate.

References

Acton, R. M., Jasny, L (2012) *An Introduction to Network Analysis with R and statnet* Sunbelt XXXII Workshop Series, March 13, 2012.

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

attribute.methods, as.network.matrix, as.sociomatrix, as.matrix.network, network.extraction

Examples

```
# read in a relational data adjacency matrix

# LOADING IN A MATRIX
## Not run:
# can download matrix file from
# https://statnet.csde.washington.edu/trac/raw-attachment/wiki/Resources/relationalData.csv
# and download vertex attribute file from
# https://statnet.csde.washington.edu/trac/raw-attachment/wiki/Resources/vertexAttributes.csv

# load in relation matrix from file
relations <- read.csv("relationalData.csv",header=FALSE,stringsAsFactors=FALSE)

# convert to matrix format from data frame
relations <- as.matrix(relations)
```

```
# load in vertex attributes
nodeInfo <- read.csv("vertexAttributes.csv",header=TRUE,stringsAsFactors=FALSE)

## End(Not run)


print(relations) # peek at matrix
print(nodeInfo)  # peek at attribute data

# Since our relational data has no row/column names, let's set them now
rownames(relations) <- nodeInfo$name
colnames(relations) <- nodeInfo$name

# create undirected network object from matrix
nrelations<-network(relations,directed=FALSE)

# it read in vertex names from matrix col names ...
network.vertex.names(nrelations)

# ATTACHING VERTEX ATTRIBUTES

# ... but could also set vertex.names with
nrelations%v%'vertex.names'<- nodeInfo$name

# load in other attributes
nrelations%v%"age" <- nodeInfo$age
nrelations%v%"sex" <- nodeInfo$sex
nrelations%v%"handed" <- nodeInfo$handed
nrelations%v%"lastDocVisit" <- nodeInfo$lastDocVisit

# Note: order of attributes in the data frame MUST match vertex ids
# otherwise the attribute will get assigned to the wrong vertex

# check that they got loaded
list.vertex.attributes(nrelations)


# what if we had an adjaceny  matrix like:
valuedMat<-matrix(c(1,2,3, 2,0,9.5,1,5,0),ncol=3,byrow=TRUE)
valuedMat

# make a network from it
valuedNet<-network(valuedMat,loops=TRUE,directed=TRUE)

# print it back out ...
as.matrix(valuedNet)

# wait, where did the values go!!?

# LOADING A MATRIX WITH VALUES

# to construct net from matrix with values:
```

```
valuedNet<-network(valuedMat,loops=TRUE,directed=TRUE,
            ignore.eval=FALSE,names.eval='myEdgeWeight')

# also have to specify the name of the attribute when converting to matrix
as.matrix(valuedNet,attrname='myEdgeWeight')

# ATTACHING EDGE ATTRIBUTES FROM A MATRIX

# maybe we have edge attributes of a different sort in another matrix like:
edgeAttrs<-matrix(c("B","Z","Q","W","A","E","L","P","A"),ncol=3,byrow=TRUE)
edgeAttrs

# we can still attach them
valuedNet<-set.edge.value(valuedNet,'someLetters',edgeAttrs)

# and extract them
as.matrix(valuedNet,attrname='someLetters')
valuedNet%e%'someLetters'

# but notice that some of the values didn't get used
# the ("A"s are missing) because there were no corresponding edges (loops)
# for the attribute to be attached to


# ATTACHING EDGE ATTRIBUTES FROM A LIST

# it is also possible to attach edge attributes directly from a list
edgeCols<-c("red","green","blue","orange","pink","brown","gray")
valuedNet<-set.edge.attribute(valuedNet,"edgeColors",edgeCols)

# but this can be risky, because we may not know the ordering of the edges,
# (especially if some have been deleted).  Does "green" go with the edge from
# 1 to 2, or from 3 to 1?

# Usually if the edge data is only availible in list form, it is safer to construct
# the network from an edgelist in the first place

# LOADING IN AN EDGELIST

# pretend we just loaded in this data.frame from a file
elData<-data.frame(
  from_id=c("1","2","3","1","3","1","2"),
  to_id=c("1", "1", "1", "2", "2", "3", "3"),
  myEdgeWeight=c(1, 2, 1, 2, 5, 3, 9.5),
  someLetters=c("B", "W", "L", "Z", "P", "Q", "E"),
  edgeCols=c("red","green","blue","orange","pink","brown","gray"),
  stringsAsFactors=FALSE
)

# peek at data
# each row corresponds to a relationship (edge) in the network
elData
```

```
# to make a network we just use the first two id columns
valuedNet2<-network(elData[,1:2],loops=TRUE)

# print it out
as.matrix(valuedNet2)

# has right edges, but no values

# to include values (with names from the columns)

valuedNet2<-network(elData,loops=TRUE)
list.edge.attributes(valuedNet2)
as.matrix(valuedNet2,attrname='someLetters')
```

`missing.edges` *Identifying and Counting Missing Edges in a Network Object*

Description

`network.naedgecount` returns the number of edges within a `network` object which are flagged as missing. The `is.na` network method returns a new network containing the missing edges.

Usage

```
## S3 method for class 'network'
is.na(x)

network.naedgecount(x, ...)
```

Arguments

`x`	an object of class `network`
`...`	additional arguments, not used

Details

The missingness of an edge is controlled by its `na` attribute (which is mandatory for all edges); `network.naedgecount` returns the number of edges for which `na==TRUE`. The `is.na` network method produces a new network object whose edges correspond to the missing (`na==TRUE`) edges of the original object, and is thus a covenient method of extracting detailed missingness information on the entire network. The network returned by `is.na` is guaranteed to have the same base network attributes (directedness, loopness, hypergraphicity, multiplexity, and bipartite constraint) as the original network object, but no other information is copied; note too that edge IDs are *not* preserved by this process (although adjacency obviously is). Since the resulting object is a `network`, standard coercion, print/summary, and other methods can be applied to it in the usual fashion.

It should be borne in mind that "missingness" in the sense used here reflects the assertion that an edge's presence or absence is unknown, *not* that said edge is known not to be present. Thus, the na count for an empty graph is properly 0, since all edges are known to be absent. Edges can be flagged as missing by setting their na attribute to TRUE using set.edge.attribute, or by appropriate use of the network assignment operators; see below for an example of the latter.

Value

is.na(x) returns a network object, and network.naedgecount(x) returns the number of missing edges.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network.edgecount, get.network.attribute, is.adjacent, is.na

Examples

```
#Create an empty network with no missing data
g<-network.initialize(5)
g[,]                        #No edges present....
network.naedgecount(g)==0   #Edges not present are not "missing"!

#Now, add some missing edges
g[1,,add.edges=TRUE]<-NA    #Establish that 1's ties are unknown
g[,]                        #Observe the missing elements
is.na(g)                    #Observe in network form
network.naedgecount(g)==4   #These elements do count!
network.edgecount(is.na(g)) #Same as above
```

---

mixingmatrix *Mixing matrix*

Description

Return the mixing matrix for a network, on a given attribute.

Usage

```
mixingmatrix(object, ...)

## S3 method for class 'network'
mixingmatrix(object, attrname, useNA = "ifany", expand.bipartite = FALSE, ...)

## S3 method for class 'mixingmatrix'
x[[...]]

## S3 method for class 'mixingmatrix'
x$name

## S3 method for class 'mixingmatrix'
is.directed(x, ...)

## S3 method for class 'mixingmatrix'
is.bipartite(x, ...)

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

Arguments

object	a network or some other data structure for which a mixing matrix is meaningful.
...	arguments passed to table.
attrname	a vertex attribute name.
useNA	one of "ifany", "no" or "always". Argument passed to table. By default (useNA = "ifany") if there are any NAs on the attribute corresponding row *and* column will be contained in the result. See Details.
expand.bipartite	logical; if object is bipartite, should we return the *square* mixing matrix representing every level of attrname against every other level, or a *rectangular* matrix considering only levels present in each bipartition?
x	mixingmatrix object
name	name of the element to extract, one of "matrix" or "type"

Details

Handling of missing values on the attribute attrname almost follows similar logic to table. If there are NAs on the attribute and useNA="ifany" (default) the result will contain both row and column for the missing values to ensure the resulting matrix is square (essentially calling table with useNA="always"). Also for that reason passing exclude parameter with NULL, NA or NaN is ignored with a warning as it may break the symmetry.

Value

Function mixingmatrix() returns an object of class mixingmatrix extending table with a cross-tabulation of edges in the object according to the values of attribute attrname for the two incident

vertices. If `object` is a *directed* network rows correspond to the "tie sender" and columns to the "tie receiver". If `object` is an *undirected* network there is no such distinction and the matrix is symmetrized. In both cases the matrix is square and all the observed values of the attribute `attrname` are represented in rows and columns. If `object` is a *bipartite* network and `expand.bipartite` is `FALSE` the resulting matrix does not have to be square as only the actually observed values of the attribute are shown for each partition, if `expand.bipartite` is `TRUE` the matrix will be square.

Functions `is.directed()` and `is.bipartite()` return `TRUE` or `FALSE`. The values will be identical for the input network `object`.

Note

The `$` and `[[` methods are included only for backward-compatiblity reason and will become defunct in future releases of the package.

Examples

```
# Interaction ties between Lake Pomona SAR organizations by sponsorship type
# of tie sender and receiver (data from Drabek et al. 1981)
data(emon)
mixingmatrix(emon$LakePomona, "Sponsorship")
```

`network` *Network Objects*

Description

Construct, coerce to, test for and print `network` objects.

Usage

```
is.network(x)

as.network(x, ...)

network(
  x,
  vertex.attr = NULL,
  vertex.attrnames = NULL,
  directed = TRUE,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE,
  ...
)

network.copy(x)
```

```
## S3 method for class 'data.frame'
as.network(
  x,
  directed = TRUE,
  vertices = NULL,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE,
  bipartite_col = "is_actor",
  ...
)

## S3 method for class 'network'
print(
  x,
  matrix.type = which.matrix.type(x),
  mixingmatrices = FALSE,
  na.omit = TRUE,
  print.adj = FALSE,
  ...
)

## S3 method for class 'network'
summary(object, na.omit = TRUE, mixingmatrices = FALSE, print.adj = TRUE, ...)
```

Arguments

x	for network, a matrix giving the network structure in adjacency, incidence, or edgelist form; otherwise, an object of class network.
...	additional arguments.
vertex.attr	optionally, a list containing vertex attributes.
vertex.attrnames	optionally, a list containing vertex attribute names.
directed	logical; should edges be interpreted as directed?
hyper	logical; are hyperedges allowed?
loops	logical; should loops be allowed?
multiple	logical; are multiplex edges allowed?
bipartite	count; should the network be interpreted as bipartite? If present (i.e., non-NULL, non-FALSE) it is the count of the number of actors in the bipartite network. In this case, the number of nodes is equal to the number of actors plus the number of events (with all actors preceeding all events). The edges are then interpreted as nondirected. Values of bipartite==0 are permited, indicating a bipartite network with zero-sized first partition.

vertices	If x is a data.frame, vertices is an optional data.frame containing the vertex attributes. The first column is assigned to the "vertex.names" and additional columns are used to set vertex attributes using their column names. If bipartite is TRUE, a logical column named "is_actor" (or the name of a column specified using the bipartite_col parameter) can be provided indicating which vertices should be considered as actors. If not provided, vertices referenced in the first column of x are assumed to be the network's actors. If your network has isolates (i.e. there are vertices referenced in vertices that are not referenced in x), the "is_actor" column is required.
bipartite_col	character(1L), default: "is_actor". The name of the logical column indicating which vertices should be considered as actors in bipartite networks.
matrix.type	one of "adjacency", "edgelist", "incidence". See edgeset.constructors for details and optional additional arguments
mixingmatrices	logical; print the mixing matrices for the discrete attributes?
na.omit	logical; omit summarization of missing attributes in network?
print.adj	logical; print the network adjacency structure?
object	an object of class network.

Details

network constructs a network class object from a matrix representation. If the matrix.type parameter is not specified, it will make a guess as to the intended edgeset.constructors function to call based on the format of these input matrices. If the class of x is not a matrix, network construction can be dispatched to other methods. For example, If the ergm package is loaded, network() can function as a shorthand for as.network.numeric with x as an integer specifying the number of nodes to be created in the random graph.

If the ergm package is loaded, network can function as a shorthand for as.network.numeric if x is an integer specifying the number of nodes. See the help page for as.network.numeric in ergm package for details.

network.copy creates a new network object which duplicates its supplied argument. (Direct assignment with <- should be used rather than network.copy in most cases.)

as.network tries to coerce its argument to a network, using the as.network.matrix functions if x is a matrix. (If the argument is already a network object, it is returned as-is and all other arguments are ignored.)

is.network tests whether its argument is a network (in the sense that it has class network).

print.network prints a network object in one of several possible formats. It also prints the list of global attributes of the network.

summary.network provides similar information.

Value

network, as.network, and print.network all return a network class object; is.network returns TRUE or FALSE.

Note

Between versions 0.5 and 1.2, direct assignment of a network object created a pointer to the original object, rather than a copy. As of version 1.2, direct assignment behaves in the same manner as `network.copy`. Direct use of the latter is thus superfluous in most situations, and is discouraged.

Many of the network package functions modify their network object arguments in-place. For example, `set.network.attribute(net,"myVal",5)` will have the same effect as `net<-set.network.attribute(net,"myVal"`. Unfortunately, the current implementation of in-place assignment breaks when the network argument is an element of a list or a named part of another object. So `set.network.attribute(myListOfNetworks[[1]],"myVa` will silently fail to modify its network argument, likely leading to incorrect output.

Author(s)

Carter T. Butts <buttsc@uci.edu> and David Hunter <dhunter@stat.psu.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network.initialize, attribute.methods, as.network.matrix, as.matrix.network, deletion.methods, edgeset.constructors, network.indicators, plot.network

Examples

```
m <- matrix(rbinom(25,1,.4),5,5)
diag(m) <- 0
g <- network(m, directed=FALSE)
summary(g)

h <- network.copy(g)       #Note: same as h<-g
summary(h)

# networks from data frames ===========================================================
#* simple networks ===================================================================
simple_edge_df <- data.frame(
  from = c("b", "c", "c", "d", "a"),
  to = c("a", "b", "a", "a", "b"),
  weight = c(1, 1, 2, 2, 3),
  stringsAsFactors = FALSE
)
simple_edge_df

as.network(simple_edge_df)

# simple networks with vertices ======================================================
simple_vertex_df <- data.frame(
  name = letters[1:5],
  residence = c("urban", "rural", "suburban", "suburban", "rural"),
```

```
  stringsAsFactors = FALSE
)
simple_vertex_df

as.network(simple_edge_df, vertices = simple_vertex_df)

as.network(simple_edge_df,
  directed = FALSE, vertices = simple_vertex_df,
  multiple = TRUE
)

#* splitting multiplex data frames into multiple networks ============================
simple_edge_df$relationship <- c(rep("friends", 3), rep("colleagues", 2))
simple_edge_df

lapply(split(simple_edge_df, f = simple_edge_df$relationship),
  as.network,
  vertices = simple_vertex_df
)

#* bipartite networks without isolates ===============================================
bip_edge_df <- data.frame(
  actor = c("a", "a", "b", "b", "c", "d", "d", "e"),
  event = c("e1", "e2", "e1", "e3", "e3", "e2", "e3", "e1"),
  actor_enjoyed_event = rep(c(TRUE, FALSE), 4),
  stringsAsFactors = FALSE
)
bip_edge_df

bip_node_df <- data.frame(
  node_id = c("a", "e1", "b", "e2", "c", "e3", "d", "e"),
  node_type = c(
    "person", "event", "person", "event", "person",
    "event", "person", "person"
  ),
  color = c(
    "red", "blue", "red", "blue", "red", "blue",
    "red", "red"
  ),
  stringsAsFactors = FALSE
)
bip_node_df

as.network(bip_edge_df, directed = FALSE, bipartite = TRUE)
as.network(bip_edge_df, directed = FALSE, vertices = bip_node_df, bipartite = TRUE)

#* bipartite networks with isolates ==================================================
bip_nodes_with_isolates <- rbind(
  bip_node_df,
  data.frame(
    node_id = c("f", "e4"),
    node_type = c("person", "event"),
    color = c("red", "blue"),
```

```
    stringsAsFactors = FALSE
  )
)
# indicate which vertices are actors via a column named `"is_actor"`
bip_nodes_with_isolates$is_actor <- bip_nodes_with_isolates$node_type == "person"
bip_nodes_with_isolates

as.network(bip_edge_df,
  directed = FALSE, vertices = bip_nodes_with_isolates,
  bipartite = TRUE
)

#* hyper networks from data frames ================================================
hyper_edge_df <- data.frame(
  from = c("a/b", "b/c", "c/d/e", "d/e"),
  to = c("c/d", "a/b/e/d", "a/b", "d/e"),
  time = 1:4,
  stringsAsFactors = FALSE
)
tibble::as_tibble(hyper_edge_df)

# split "from" and "to" at `"/"`, coercing them to list columns
hyper_edge_df$from <- strsplit(hyper_edge_df$from, split = "/")
hyper_edge_df$to <- strsplit(hyper_edge_df$to, split = "/")
tibble::as_tibble(hyper_edge_df)

as.network(hyper_edge_df,
  directed = FALSE, vertices = simple_vertex_df,
  hyper = TRUE, loops = TRUE
)

# convert network objects back to data frames ====================================
simple_g <- as.network(simple_edge_df, vertices = simple_vertex_df)
as.data.frame(simple_g)
as.data.frame(simple_g, unit = "vertices")

bip_g <- as.network(bip_edge_df,
  directed = FALSE, vertices = bip_node_df,
  bipartite = TRUE
)
as.data.frame(bip_g)
as.data.frame(bip_g, unit = "vertices")

hyper_g <- as.network(hyper_edge_df,
  directed = FALSE, vertices = simple_vertex_df,
  hyper = TRUE, loops = TRUE
)
as.data.frame(hyper_g)
as.data.frame(hyper_g, unit = "vertices")
```

---

network.arrow *Add Arrows or Segments to a Plot*

Description

network.arrow draws a segment or arrow between two pairs of points; unlike arrows or segments, the new plot element is drawn as a polygon.

Usage

```
network.arrow(
  x0,
  y0,
  x1,
  y1,
  length = 0.1,
  angle = 20,
  width = 0.01,
  col = 1,
  border = 1,
  lty = 1,
  offset.head = 0,
  offset.tail = 0,
  arrowhead = TRUE,
  curve = 0,
  edge.steps = 50,
  ...
)
```

Arguments

x0	A vector of x coordinates for points of origin
y0	A vector of y coordinates for points of origin
x1	A vector of x coordinates for destination points
y1	A vector of y coordinates for destination points
length	Arrowhead length, in current plotting units
angle	Arrowhead angle (in degrees)
width	Width for arrow body, in current plotting units (can be a vector)
col	Arrow body color (can be a vector)
border	Arrow border color (can be a vector)
lty	Arrow border line type (can be a vector)
offset.head	Offset for destination point (can be a vector)
offset.tail	Offset for origin point (can be a vector)
arrowhead	Boolean; should arrowheads be used? (Can be a vector))
curve	Degree of edge curvature (if any), in current plotting units (can be a vector)
edge.steps	For curved edges, the number of steps to use in approximating the curve (can be a vector)
...	Additional arguments to polygon

Details

network.arrow provides a useful extension of segments and arrows when fine control is needed over the resulting display. (The results also look better.) Note that edge curvature is quadratic, with curve providing the maximum horizontal deviation of the edge (left-handed). Head/tail offsets are used to adjust the end/start points of an edge, relative to the baseline coordinates; these are useful for functions like plot.network, which need to draw edges incident to vertices of varying radii.

Value

None.

Note

network.arrow is a direct adaptation of gplot.arrow from the sna package.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

plot.network, network.loop, polygon

Examples

```
#Plot two points
plot(1:2,1:2)

#Add an edge
network.arrow(1,1,2,2,width=0.01,col="red",border="black")
```

---

network.density *Compute the Density of a Network*

Description

network.density computes the density of its argument.

Usage

```
network.density(x, na.omit = TRUE, discount.bipartite = FALSE)
```

Arguments

x	an object of class network
na.omit	logical; omit missing edges from extant edges when assessing density?
discount.bipartite	logical; if x is bipartite, should "forbidden" edges be excluded from the count of potential edges?

Details

The density of a network is defined as the ratio of extant edges to potential edges. We do not currently consider edge values; missing edges are omitted from extent (but not potential) edge count when na.omit==TRUE.

Value

The network density.

Warning

network.density relies on network attributes (see network.indicators) to determine the properties of the underlying network object. If these are set incorrectly (e.g., multiple edges in a non-multiplex network, network coded with directed edges but set to "undirected", etc.), surprising results may ensue.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

network.edgecount, network.size

Examples

```
#Create an arbitrary adjacency matrix
m<-matrix(rbinom(25,1,0.5),5,5)
diag(m)<-0

g<-network.initialize(5)    #Initialize the network
network.density(g)          #Calculate the density
```

`network.dyadcount` *Return the Number of (Possibly Directed) Dyads in a Network Object*

Description

`network.dyadcount` returns the number of possible dyads within a `network`, removing those flagged as missing if desired. If the network is directed, directed dyads are counted accordingly.

Usage

```
## S3 method for class 'network'
network.dyadcount(x, na.omit = TRUE, ...)
```

Arguments

`x`	an object of class `network`
`na.omit`	logical; omit edges with `na==TRUE` from the count?
`...`	possible additional arguments, used by other implementations

Details

The return value `network.dyadcount` is equal to the number of dyads, minus the number of `NULL` edges (and missing edges, if `na.omit==TRUE`). If `x` is directed, the number of directed dyads is returned. If the network allows loops, the number of possible entries on the diagnonal is added. Allthough the function does not give an error on multiplex networks or hypergraphs, the results probably don't make sense.

Value

The number of dyads in the network

Author(s)

Mark S. Handcock <handcock@stat.washington.edu>, skyebend

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`get.network.attribute`, `network.edgecount`, `is.directed`

Examples

```
#Create a directed network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)
network.dyadcount(g)==6                #Verify the directed dyad count
g<-network(m|t(m),directed=FALSE)
network.dyadcount(g)==3                        #nC2 in undirected case
```

---

network.edgecount *Return the Number of Edges in a Network Object*

Description

network.edgecount returns the number of edges within a network, removing those flagged as missing if desired.

Usage

```
## S3 method for class 'network'
network.edgecount(x, na.omit = TRUE, ...)
```

Arguments

x	an object of class network
na.omit	logical; omit edges with na==TRUE from the count?
...	additional arguments, used by extending functio

Details

The return value is the number of distinct edges within the network object, including multiplex edges as appropriate. (So if there are 3 edges from vertex i to vertex j, each contributes to the total edge count.)

The return value network.edgecount is in the present implementation related to the (required) mnext network attribute. mnext is an internal legacy attribute that currently indicates the index number of the next edge to be added to a network object. (Do not modify it unless you enjoy unfortunate surprises.) The number of edges returned by network.edgecount is equal to x%n%"mnext"-1, minus the number of NULL edges (and missing edges, if na.omit==TRUE). Note that g%n%"mnext"-1 cannot, by itself, be counted upon to be an accurate count of the number of edges! As mnext is not part of the API (and is not guaranteed to remain), users and developers are urged to use network.edgecount instead.

Value

The number of edges

Warning

network.edgecount uses the real state of the network object to count edges, not the state it hypothetically should have. Thus, if you add extra edges to a non-multiplex network, directed edges to an undirected network, etc., the actual number of edges in the object will be returned (and not the number you would expect if you relied only on the putative number of possible edges as reflected by the network.indicators). Don't create network objects with contradictory attributes unless you know what you are doing.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

get.network.attribute

Examples

```
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-1
g<-network(m)
network.edgecount(g)==3   #Verify the edgecount
```

---

network.edgelabel *Plots a label corresponding to an edge in a network plot.*

Description

Draws a text labels on (or adjacent to) the line segments connecting vertices on a network plot.

Usage

```
network.edgelabel(
  px0,
  py0,
  px1,
  py1,
  label,
  directed,
  loops = FALSE,
```

```
  cex,
  curve = 0,
  ...
)
```

Arguments

px0	vector of x coordinates of tail vertex of the edge
py0	vector of y coordinates of tail vertex of the edge
px1	vector of x coordinates of head vertex of the edge
py1	vector of y coordinate of head vertex of the edge
label	vector strings giving labels to be drawn for edge edge
directed	logical: is the underlying network directed? If FALSE, labels will be drawn in the middle of the line segment, otherwise in the first 3rd so that the labels for edges pointing in the opposite direction will not overlap.
loops	logical: if true, assuming the labels to be drawn belong to loop-type edges and render appropriately
cex	numeric vector giving the text expansion factor for each label
curve	numeric vector controling the extent of edge curvature (0 = straight line edges)
...	additional arguments to be passed to text

Details

Called internally by plot.network when edge.label parameter is used. For directed, non-curved edges, the labels are shifted towards the tail of the edge. Labels for curved edges are not shifted because opposite-direction edges curve the opposite way. Makes a crude attempt to shift labels to either side of line, and to draw the edge labels for self-loops near the vertex. No attempt is made to avoid overlap between vertex and edge labels.

Value

no value is returned but text will be rendered on the active plot

Author(s)

skyebend

network.extraction *Extraction and Replacement Operators for Network Objects*

Description

Various operators which allow extraction or replacement of various components of a network object.

Usage

```
## S3 method for class 'network'
x[i, j, na.omit = FALSE]

## S3 replacement method for class 'network'
x[i, j, names.eval = NULL, add.edges = FALSE] <- value

x %e% attrname

x %e% attrname <- value

x %eattr% attrname

x %eattr% attrname <- value

x %n% attrname

x %n% attrname <- value

x %nattr% attrname

x %nattr% attrname <- value

x %v% attrname

x %v% attrname <- value

x %vattr% attrname

x %vattr% attrname <- value
```

Arguments

x	an object of class network.
i, j	indices of the vertices with respect to which adjacency is to be tested. Empty values indicate that all vertices should be employed (see below).
na.omit	logical; should missing edges be omitted (treated as no-adjacency), or should NAs be returned? (Default: return NA on missing.)

names.eval	optionally, the name of an edge attribute to use for assigning edge values.
add.edges	logical; should new edges be added to x where edges are absent and the appropriate element of value is non-zero?
value	the value (or set thereof) to be assigned to the selected element of x.
attrname	the name of a network or vertex attribute (as appropriate).

Details

Indexing for edge extraction operates in a manner analogous to matrix objects. Thus, x[,] selects all vertex pairs, x[1,-5] selects the pairing of vertex 1 with all vertices except for 5, etc. Following this, it is acceptable for i and/or j to be logical vectors indicating which vertices are to be included. During assignment, an attempt is made to match the elements of value to the extracted pairs in an intelligent way; in particular, elements of value will be replicated if too few are supplied (allowing expressions like x[1,]<-1). Where names.eval==NULL, zero and non-zero values are taken to indicate the presence of absence of edges. x[2,4]<-6 thus adds a single (2,4) edge to x, and x[2,4]<-0 removes such an edge (if present). If x is multiplex, assigning 0 to a vertex pair will eliminate *all* edges on that pair. Pairs are taken to be directed where is.directed(x)==TRUE, and undirected where is.directed(x)==FALSE.

If an edge attribute is specified using names.eval, then the provided values will be assigned to that attribute. When assigning values, only extant edges are employed (unless add.edges==TRUE); in the latter case, any non-zero assignment results in the addition of an edge where currently absent. If the attribute specified is not present on a given edge, it is added. Otherwise, any existing value is overwritten. The %e% operator can also be used to extract/assign edge values; in those roles, it is respectively equivalent to get.edge.value(x,attrname) and set.edge.value(x,attrname=attrname,value=value) (if value is a matrix) and set.edge.attribute(x,attrname=attrname,value=value) (if value is anything else). That is, if value is a matrix, the assignment operator treats it as an adjacency matrix; and if not, it treats it as a vector (recycled as needed) in the internal ordering of edges (i.e., edge IDs), skipping over deleted edges. In no case will attributes be assigned to nonexisted edges.

The %n% and %v% operators serve as front-ends to the network and vertex extraction/assignment functions (respectively). In the extraction case, x %n% attrname is equivalent to get.network.attribute(x,attrname), with x %v% attrname corresponding to get.vertex.attribute(x,attrname). In assignment, the respective equivalences are to set.network.attribute(x,attrname,value) and set.vertex.attribute(x,attrname,v
Note that the %% assignment forms are generally slower than the named versions of the functions beause they will trigger an additional internal copy of the network object.

The %eattr%, %nattr%, and %vattr% operators are equivalent to %e%, %n%, and %v% (respectively). The short forms are more succinct, but may produce less readable code.

Value

The extracted data, or none.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

is.adjacent, as.sociomatrix, attribute.methods, add.edges, network.operators, and get.inducedSubgraph

Examples

```
#Create a random graph (inefficiently)
g<-network.initialize(10)
g[,]<-matrix(rbinom(100,1,0.1),10,10)
plot(g)

#Demonstrate edge addition/deletion
g[,]<-0
g[1,]<-1
g[2:3,6:7]<-1
g[,]

#Set edge values
g[,,names.eval="boo"]<-5
as.sociomatrix(g,"boo")
#Assign edge values from a vector
g %e% "hoo" <- "wah"
g %e% "hoo"
g %e% "om" <- c("wow","whee")
g %e% "om"
#Assign edge values as a sociomatrix
g %e% "age" <- matrix(1:100, 10, 10)
g %e% "age"
as.sociomatrix(g,"age")

#Set/retrieve network and vertex attributes
g %n% "blah" <- "Pork!"                #The other white meat?
g %n% "blah" == "Pork!"                #TRUE!
g %v% "foo" <- letters[10:1]           #Letter the vertices
g %v% "foo" == letters[10:1]           #All TRUE
```

---

network.indicators *Indicator Functions for Network Properties*

Description

Various indicators for properties of network class objects.

Usage

```
has.loops(x)

is.bipartite(x, ...)

## S3 method for class 'network'
is.bipartite(x, ...)

is.directed(x, ...)

## S3 method for class 'network'
is.directed(x, ...)

is.hyper(x)

is.multiplex(x)
```

Arguments

x	an object of class network
...	other arguments passed to/from other methods

Details

These methods are the standard means of assessing the state of a network object; other methods can (and should) use these routines in governing their own behavior. As such, improper setting of the associated attributes may result in unpleasantly creative results. (See the edge.check argument to add.edges for an example of code which makes use of these network properties.)

The functions themselves behave has follows:

has.loops returns TRUE iff x is allowed to contain loops (or loop-like edges, in the hypergraphic case).

is.bipartite returns TRUE iff the x has been explicitly bipartite-coded. Values of bipartite=NULL, and bipartite=FALSE will evaluate to FALSE, numeric values of bipartite>=0 will evaluate to TRUE. (The value bipartite==0 indicates that it is a bipartite network with a zero-sized first partition.) Note that is.bipartite refers only to the storage properties of x and how it should be treated by some algorithms; is.bipartite(x)==FALSE it does *not* mean that x cannot admit a bipartition!

is.directed returns TRUE iff the edges of x are to be interpreted as directed.

is.hyper returns TRUE iff x is allowed to contain hypergraphic edges.

is.multiplex returns TRUE iff x is allowed to contain multiplex edges.

Value

TRUE or FALSE

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network, get.network.attribute, set.network.attribute, add.edges

Examples

```
g<-network.initialize(5)    #Initialize the network
is.bipartite(g)
is.directed(g)
is.hyper(g)
is.multiplex(g)
has.loops(g)
```

---

network.initialize *Initialize a Network Class Object*

Description

Create and initialize a network object with n vertices.

Usage

```
network.initialize(
  n,
  directed = TRUE,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE
)
```

Arguments

n	the number of vertices to initialize
directed	logical; should edges be interpreted as directed?
hyper	logical; are hyperedges allowed?
loops	logical; should loops be allowed?
multiple	logical; are multiplex edges allowed?

bipartite — count; should the network be interpreted as bipartite? If present (i.e., non-NULL) it is the count of the number of actors in the first mode of the bipartite network. In this case, the overall number of vertices is equal to the number of 'actors' (first mode) plus the number of 'events' (second mode), with the vertex.ids of all actors preceeding all events. The edges are then interpreted as nondirected.

Details

Generally, `network.initialize` is called by other constructor functions as part of the process of creating a network.

Value

An object of class `network`

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`network`, `as.network.matrix`

Examples

```
g<-network.initialize(5)  #Create an empty graph on 5 vertices
```

---

network.layout *Vertex Layout Functions for plot.network*

Description

Various functions which generate vertex layouts for the `plot.network` visualization routine.

Usage

```
network.layout.circle(nw, layout.par)

network.layout.fruchtermanreingold(nw, layout.par)

network.layout.kamadakawai(nw, layout.par)
```

Arguments

nw	a network object, as passed by plot.network.
layout.par	a list of parameters.

Details

Vertex layouts for network visualization pose a difficult problem – there is no single, "good" layout algorithm, and many different approaches may be valuable under different circumstances. With this in mind, plot.network allows for the use of arbitrary vertex layout algorithms via the network.layout.* family of routines. When called, plot.network searches for a network.layout function whose fourth name matches its mode argument (see plot.network help for more information); this function is then used to generate the layout for the resulting plot. In addition to the routines documented here, users may add their own layout functions as needed. The requirements for a network.layout function are as follows:

1. the first argument, nw, must be a network object;
2. the second argument, layout.par, must be a list of parameters (or NULL, if no parameters are specified); and
3. the return value must be a real matrix of dimension c(2,network.size(nw)), whose rows contain the vertex coordinates.

Other than this, anything goes. (In particular, note that layout.par could be used to pass additional matrices or other information, if needed. Alternately, it is possible to make layout methods that respond to covariates on the network object, which are maintained intact by plot.network.)

The network.layout functions currently supplied by default are as follows (with n==network.size(nw)):

circle This function places vertices uniformly in a circle; it takes no arguments.

fruchtermanreingold This function generates a layout using a variant of Fruchterman and Reingold's force-directed placement algorithm. It takes the following arguments:

layout.par$niter This argument controls the number of iterations to be employed. Larger values take longer, but will provide a more refined layout. (Defaults to 500.)

layout.par$max.delta Sets the maximum change in position for any given iteration. (Defaults to n.)

layout.par$area Sets the "area" parameter for the F-R algorithm. (Defaults to n^2.)

layout.par$cool.exp Sets the cooling exponent for the annealer. (Defaults to 3.)

layout.par$repulse.rad Determines the radius at which vertex-vertex repulsion cancels out attraction of adjacent vertices. (Defaults to area*log(n).)

layout.par$ncell To speed calculations on large graphs, the plot region is divided at each iteration into ncell by ncell "cells", which are used to define neighborhoods for force calculation. Moderate numbers of cells result in fastest performance; too few cells (down to 1, which produces "pure" F-R results) can yield odd layouts, while too many will result in long layout times. (Defaults to n^0.4.)

layout.par$cell.jitter Jitter factor (in units of cell width) used in assigning vertices to cells. Small values may generate "grid-like" anomalies for graphs with many isolates. (Defaults to 0.5.)

layout.par$cell.pointpointrad Squared "radius" (in units of cells) such that exact point interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart. Higher values approximate the true F-R solution, but increase computational cost. (Defaults to `0`.)

layout.par$cell.pointcellrad Squared "radius" (in units of cells) such that approximate point/cell interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart (and not within the point/point radius). Higher values provide somewhat better approximations to the true F-R solution at slightly increased computational cost. (Defaults to `18`.)

layout.par$cell.cellcellrad Squared "radius" (in units of cells) such that approximate cell/cell interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart (and not within the point/point or point/cell radii). Higher values provide somewhat better approximations to the true F-R solution at slightly increased computational cost. Note that cells beyond this radius (if any) do not interact, save through edge attraction. (Defaults to `ncell^2`.)

layout.par$seed.coord A two-column matrix of initial vertex coordinates. (Defaults to a random circular layout.)

kamadakawai This function generates a vertex layout using a version of the Kamada-Kawai force-directed placement algorithm. It takes the following arguments:

layout.par$niter This argument controls the number of iterations to be employed. (Defaults to 1000.)

layout.par$sigma Sets the base standard deviation of position change proposals. (Defaults to `n/4`.)

layout.par$initemp Sets the initial "temperature" for the annealing algorithm. (Defaults to 10.)

layout.par$cool.exp Sets the cooling exponent for the annealer. (Defaults to 0.99.)

layout.par$kkconst Sets the Kamada-Kawai vertex attraction constant. (Defaults to `n)^2`.)

layout.par$elen Provides the matrix of interpoint distances to be approximated. (Defaults to the geodesic distances of `nw` after symmetrizing, capped at `sqrt(n)`.)

layout.par$seed.coord A two-column matrix of initial vertex coordinates. (Defaults to a gaussian layout.)

Value

A matrix whose rows contain the x,y coordinates of the vertices of `d`.

Note

The `network.layout` routines shown here are adapted directly from the `gplot.layout` routines of the `sna` package.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Fruchterman, T.M.J. and Reingold, E.M. (1991). "Graph Drawing by Force-directed Placement." *Software - Practice and Experience,* 21(11):1129-1164.

Kamada, T. and Kawai, S. (1989). "An Algorithm for Drawing General Undirected Graphs." *Information Processing Letters,* 31(1):7-15.

See Also

plot.network

`network.loop` *Add Loops to a Plot*

Description

`network.loop` draws a "loop" at a specified location; this is used to designate self-ties in `plot.network`.

Usage

```
network.loop(
  x0,
  y0,
  length = 0.1,
  angle = 10,
  width = 0.01,
  col = 1,
  border = 1,
  lty = 1,
  offset = 0,
  edge.steps = 10,
  radius = 1,
  arrowhead = TRUE,
  xctr = 0,
  yctr = 0,
  ...
)
```

Arguments

`x0`	a vector of x coordinates for points of origin.
`y0`	a vector of y coordinates for points of origin.
`length`	arrowhead length, in current plotting units.
`angle`	arrowhead angle (in degrees).

width	width for loop body, in current plotting units (can be a vector).
col	loop body color (can be a vector).
border	loop border color (can be a vector).
lty	loop border line type (can be a vector).
offset	offset for origin point (can be a vector).
edge.steps	number of steps to use in approximating curves.
radius	loop radius (can be a vector).
arrowhead	boolean; should arrowheads be used? (Can be a vector.)
xctr	x coordinate for the central location away from which loops should be oriented.
yctr	y coordinate for the central location away from which loops should be oriented.
...	additional arguments to polygon.

Details

network.loop is the companion to network.arrow; like the latter, plot elements produced by network.loop are drawn using polygon, and as such are scaled based on the current plotting device. By default, loops are drawn so as to encompass a circular region of radius radius, whose center is offset units from x0,y0 and at maximum distance from xctr,yctr. This is useful for functions like plot.network, which need to draw loops incident to vertices of varying radii.

Value

None.

Note

network.loop is a direct adaptation of gplot.loop, from the sna package.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

network.arrow, plot.network, polygon

Examples

```
#Plot a few polygons with loops
plot(0,0,type="n",xlim=c(-2,2),ylim=c(-2,2),asp=1)
network.loop(c(0,0),c(1,-1),col=c(3,2),width=0.05,length=0.4,
  offset=sqrt(2)/4,angle=20,radius=0.5,edge.steps=50,arrowhead=TRUE)
polygon(c(0.25,-0.25,-0.25,0.25,NA,0.25,-0.25,-0.25,0.25),
    c(1.25,1.25,0.75,0.75,NA,-1.25,-1.25,-0.75,-0.75),col=c(2,3))
```

network.operators *Network Operators*

Description

These operators allow for algebraic manipulation of relational structures.

Usage

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

## S3 method for class 'network'
e1 - e2

## S3 method for class 'network'
e1 * e2

## S3 method for class 'network'
!e1

## S3 method for class 'network'
e1 | e2

## S3 method for class 'network'
e1 & e2

## S3 method for class 'network'
e1 %c% e2
```

Arguments

e1	an object of class network.
e2	another network.

Details

In general, the binary network operators function by producing a new network object whose edge structure is based on that of the input networks. The properties of the new structure depend upon the inputs as follows:

- The size of the new network is equal to the size of the input networks (for all operators save %c%), which must themselves be of equal size. Likewise, the bipartite attributes of the inputs must match, and this is preserved in the output.
- If either input network allows loops, multiplex edges, or hyperedges, the output acquires this property. (If both input networks do not allow these features, then the features are disallowed in the output network.)

- If either input network is directed, the output is directed; if exactly one input network is directed, the undirected input is treated as if it were a directed network in which all edges are reciprocated.
- Supplemental attributes (including vertex names, but not edgwise missingness) are not transferred to the output.

The unary operator acts per the above, but with a single input. Thus, the output network has the same properties as the input, with the exception of supplemental attributes.

The behavior of the composition operator, `%c%`, is somewhat more complex than the others. In particular, it will return a bipartite network whenever either input network is bipartite *or* the vertex names of the two input networks do not match (or are missing). If both inputs are non-bipartite and have identical vertex names, the return value will have the same structure (but with loops). This behavior corresponds to the interpretation of the composition operator as counting walks on labeled sets of vertices.

Hypergraphs are not yet supported by these routines, but ultimately will be (as suggested by the above).

The specific operations carried out by these operators are generally self-explanatory in the non-multiplex case, but semantics in the latter circumstance bear elaboration. The following summarizes the behavior of each operator:

`+` An (i, j) edge is created in the return graph for every (i, j) edge in each of the input graphs.

`-` An (i, j) edge is created in the return graph for every (i, j) edge in the first input that is not matched by an (i, j) edge in the second input; if the second input has more (i, j) edges than the first, no (i, j) edges are created in the return graph.

`*` An (i, j) edge is created for every pairing of (i, j) edges in the respective input graphs.

`%c%` An (i, j) edge is created in the return graph for every edge pair $(i, k), (k, j)$ with the first edge in the first input and the second edge in the second input.

`!` An (i, j) edge is created in the return graph for every (i, j) in the input not having an edge.

`|` An (i, j) edge is created in the return graph if either input contains an (i, j) edge.

`&` An (i, j) edge is created in the return graph if both inputs contain an (i, j) edge.

Semantics for missing-edge cases follow from the above, under the interpretation that edges with `na==TRUE` are viewed as having an unknown state. Thus, for instance, `x*y` with `x` having 2 (i, j) non-missing and 1 missing edge and `y` having 3 respective non-missing and 2 missing edges will yield an output network with 6 non-missing and 9 missing (i, j) edges.

Value

The resulting network.

Note

Currently, there is a naming conflict between the composition operator and the `%c%` operator in the sna package. This will be resolved in future releases; for the time being, one can determine which version of `%c%` is in use by varying which package is loaded first.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: University of Cambridge Press.

See Also

network.extraction

Examples

```
#Create an in-star
m<-matrix(0,6,6)
m[2:6,1]<-1
g<-network(m)
plot(g)

#Compose g with its transpose
gcgt<-g %c% (network(t(m)))
plot(gcgt)
gcgt

#Show the complement of g
!g

#Perform various arithmatic and logical operations
(g+gcgt)[,] == (g|gcgt)[,]             #All TRUE
(g-gcgt)[,] == (g&(!(gcgt)))[,]
(g*gcgt)[,] == (g&gcgt)[,]
```

---

network.size *Return the Size of a Network*

Description

network.size returns the order of its argument (i.e., number of vertices).

Usage

```
network.size(x, ...)
```

Arguments

x	an object of class network
...	additional arguments, not used

Details

network.size(x) is equivalent to get.network.attribute(x,"n"); the function exists as a convenience.

Value

The network size

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

get.network.attribute

Examples

```
#Initialize a network
g<-network.initialize(7)
network.size(g)
```

---

network.vertex *Add Vertices to a Plot*

Description

network.vertex adds one or more vertices (drawn using polygon) to a plot.

Usage

```
network.vertex(
  x,
  y,
  radius = 1,
  sides = 4,
  border = 1,
  col = 2,
  lty = NULL,
  rot = 0,
  lwd = 1,
  ...
)
```

Arguments

x	a vector of x coordinates.
y	a vector of y coordinates.
radius	a vector of vertex radii.
sides	a vector containing the number of sides to draw for each vertex.
border	a vector of vertex border colors.
col	a vector of vertex interior colors.
lty	a vector of vertex border line types.
rot	a vector of vertex rotation angles (in degrees).
lwd	a vector of vertex border line widths.
...	Additional arguments to polygon

Details

network.vertex draws regular polygons of specified radius and number of sides, at the given coordinates. This is useful for routines such as plot.network, which use such shapes to depict vertices.

Value

None

Note

network.vertex is a direct adaptation of gplot.vertex from the sna package.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

plot.network, polygon

Examples

```
#Open a plot window, and place some vertices
plot(0,0,type="n",xlim=c(-1.5,1.5),ylim=c(-1.5,1.5),asp=1)
network.vertex(cos((1:10)/10*2*pi),sin((1:10)/10*2*pi),col=1:10,
    sides=3:12,radius=0.1)
```

---

permute.vertexIDs *Permute (Relabel) the Vertices Within a Network*

Description

permute.vertexIDs permutes the vertices within a given network in the specified fashion. Since this occurs internally (at the level of vertex IDs), it is rarely of interest to end-users.

Usage

```
permute.vertexIDs(x, vids)
```

Arguments

x	an object of class network.
vids	a vector of vertex IDs, in the order to which they are to be permuted.

Details

permute.vertexIDs alters the internal ordering of vertices within a network. For most practical applications, this should not be necessary – de facto permutation can be accomplished by altering the appropriate vertex attributes. permute.vertexIDs is needed for certain other routines (such as delete.vertices), where it is used in various arcane and ineffable ways.

Value

Invisibly, a pointer to the permuted network. permute.vertexIDs modifies its argument in place.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network

Examples

```
data(flo)                    #Load the Florentine Families data
nflo<-network(flo)                 #Create a network object
n<-network.size(nflo)             #Get the number of vertices
permute.vertexIDs(nflo,n:1)             #Reverse the vertices
all(flo[n:1,n:1]==as.sociomatrix(nflo))       #Should be TRUE
```

---

plot.network.default *Two-Dimensional Visualization for Network Objects*

Description

`plot.network` produces a simple two-dimensional plot of network `x`, using optional attribute `attrname` to set edge values. A variety of options are available to control vertex placement, display details, color, etc.

Usage

```
## S3 method for class 'network'
plot(x, ...)

## Default S3 method:
plot.network(x, attrname = NULL,
   label = network.vertex.names(x), coord = NULL, jitter = TRUE,
   thresh = 0, usearrows = TRUE, mode = "fruchtermanreingold",
   displayisolates = TRUE, interactive = FALSE, xlab = NULL,
   ylab = NULL, xlim = NULL, ylim = NULL, pad = 0.2, label.pad = 0.5,
   displaylabels = !missing(label), boxed.labels = FALSE, label.pos = 0,
   label.bg = "white", vertex.sides = 50, vertex.rot = 0, vertex.lwd=1,
   arrowhead.cex = 1, label.cex = 1, loop.cex = 1, vertex.cex = 1,
   edge.col = 1, label.col = 1, vertex.col = 2, label.border = 1,
   vertex.border = 1, edge.lty = 1, label.lty = NULL, vertex.lty = 1,
   edge.lwd = 0, edge.label = NULL, edge.label.cex = 1,
```

```
edge.label.col = 1, label.lwd = par("lwd"), edge.len = 0.5,
edge.curve = 0.1, edge.steps = 50, loop.steps = 20,
object.scale = 0.01, uselen = FALSE, usecurve = FALSE,
suppress.axes = TRUE, vertices.last = TRUE, new = TRUE,
layout.par = NULL, ...)
```

Arguments

x	an object of class network.
...	additional arguments to plot.
attrname	an optional edge attribute, to be used to set edge values.
label	a vector of vertex labels, if desired; defaults to the vertex labels returned by network.vertex.names. If label has one element and it matches with a vertex attribute name, the value of the attribute will be used. Note that labels may be set but hidden by the displaylabels argument.
coord	user-specified vertex coordinates, in an network.size(x)x2 matrix. Where this is specified, it will override the mode setting.
jitter	boolean; should the output be jittered?
thresh	real number indicating the lower threshold for tie values. Only ties of value >thresh are displayed. By default, thresh=0.
usearrows	boolean; should arrows (rather than line segments) be used to indicate edges?
mode	the vertex placement algorithm; this must correspond to a network.layout function.
displayisolates	boolean; should isolates be displayed?
interactive	boolean; should interactive adjustment of vertex placement be attempted?
xlab	x axis label.
ylab	y axis label.
xlim	the x limits (min, max) of the plot.
ylim	the y limits of the plot.
pad	amount to pad the plotting range; useful if labels are being clipped.
label.pad	amount to pad label boxes (if boxed.labels==TRUE), in character size units.
displaylabels	boolean; should vertex labels be displayed?
boxed.labels	boolean; place vertex labels within boxes?
label.pos	position at which labels should be placed, relative to vertices. 0 results in labels which are placed away from the center of the plotting region; 1, 2, 3, and 4 result in labels being placed below, to the left of, above, and to the right of vertices (respectively); and label.pos>=5 results in labels which are plotted with no offset (i.e., at the vertex positions).
label.bg	background color for label boxes (if boxed.labels==TRUE); may be a vector, if boxes are to be of different colors.

vertex.sides	number of polygon sides for vertices; may be given as a vector or a vertex attribute name, if vertices are to be of different types. As of v1.12, radius of polygons are scaled so that all shapes have equal area
vertex.rot	angle of rotation for vertices (in degrees); may be given as a vector or a vertex attribute name, if vertices are to be rotated differently.
vertex.lwd	line width of vertex borders; may be given as a vector or a vertex attribute name, if vertex borders are to have different line widths.
arrowhead.cex	expansion factor for edge arrowheads.
label.cex	character expansion factor for label text.
loop.cex	expansion factor for loops; may be given as a vector or a vertex attribute name, if loops are to be of different sizes.
vertex.cex	expansion factor for vertices; may be given as a vector or a vertex attribute name, if vertices are to be of different sizes.
edge.col	color for edges; may be given as a vector, adjacency matrix, or edge attribute name, if edges are to be of different colors.
label.col	color for vertex labels; may be given as a vector or a vertex attribute name, if labels are to be of different colors.
vertex.col	color for vertices; may be given as a vector or a vertex attribute name, if vertices are to be of different colors.
label.border	label border colors (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different colors.
vertex.border	border color for vertices; may be given as a vector or a vertex attribute name, if vertex borders are to be of different colors.
edge.lty	line type for edge borders; may be given as a vector, adjacency matrix, or edge attribute name, if edge borders are to have different line types.
label.lty	line type for label boxes (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different line types.
vertex.lty	line type for vertex borders; may be given as a vector or a vertex attribute name, if vertex borders are to have different line types.
edge.lwd	line width scale for edges; if set greater than 0, edge widths are scaled by edge.lwd*dat. May be given as a vector, adjacency matrix, or edge attribute name, if edges are to have different line widths.
edge.label	if non-NULL, labels for edges will be drawn. May be given as a vector, adjacency matrix, or edge attribute name, if edges are to have different labels. A single value of TRUE will use edge ids as labels. NOTE: currently doesn't work for curved edges.
edge.label.cex	character expansion factor for edge label text; may be given as a vector or a edge attribute name, if edge labels are to have different sizes.
edge.label.col	color for edge labels; may be given as a vector or a edge attribute name, if labels are to be of different colors.
label.lwd	line width for label boxes (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different line widths.

edge.len	if uselen==TRUE, curved edge lengths are scaled by edge.len.
edge.curve	if usecurve==TRUE, the extent of edge curvature is controlled by edge.curv. May be given as a fixed value, vector, adjacency matrix, or edge attribute name, if edges are to have different levels of curvature.
edge.steps	for curved edges (excluding loops), the number of line segments to use for the curve approximation.
loop.steps	for loops, the number of line segments to use for the curve approximation.
object.scale	base length for plotting objects, as a fraction of the linear scale of the plotting region. Defaults to 0.01.
uselen	boolean; should we use edge.len to rescale edge lengths?
usecurve	boolean; should we use edge.curve?
suppress.axes	boolean; suppress plotting of axes?
vertices.last	boolean; plot vertices after plotting edges?
new	boolean; create a new plot? If new==FALSE, vertices and edges will be added to the existing plot.
layout.par	parameters to the network.layout function specified in mode.

Details

plot.network is the standard visualization tool for the network class. By means of clever selection of display parameters, a fair amount of display flexibility can be obtained. Vertex layout – if not specified directly using coord – is determined via one of the various available algorithms. These should be specified via the mode argument; see network.layout for a full list. User-supplied layout functions are also possible – see the aforementioned man page for details.

Note that where is.hyper(x)==TRUE, the network is converted to bipartite adjacency form prior to computing coordinates. If interactive==TRUE, then the user may modify the initial network layout by selecting an individual vertex and then clicking on the location to which this vertex is to be moved; this process may be repeated until the layout is satisfactory.

Value

A two-column matrix containing the vertex positions as x,y coordinates

Note

plot.network is adapted (with minor modifications) from the gplot function of the sna library (authors: Carter T. Butts and Alex Montgomery); eventually, these two packages will be integrated.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

network, network.arrow, network.loop, network.vertex

Examples

```
#Construct a sparse graph
m<-matrix(rbinom(100,1,1.5/9),10)
diag(m)<-0
g<-network(m)

#Plot the graph
plot(g)

#Load Padgett's marriage data
data(flo)
nflo<-network(flo)
#Display the network, indicating degree and flagging the Medicis
plot(nflo, vertex.cex=apply(flo,2,sum)+1, usearrows=FALSE,
    vertex.sides=3+apply(flo,2,sum),
    vertex.col=2+(network.vertex.names(nflo)=="Medici"))
```

---

plotArgs.network *Expand and transform attributes of networks to values appropriate for aguments to plot.network*

Description

This is primairly an internal function called by plot.network or by external packages such as ndtv that want to prepare plot.network graphic arguments in a standardized way.

Usage

```
plotArgs.network(x, argName, argValue, d = NULL, edgetouse = NULL)
```

Arguments

x	a network object which is going to be plotted
argName	character, the name of plot.network graphic parameter
argValue	value for the graphic paramter named in argName which to be transformed/prepared. For many attributes, if this is a single character vector it will be assumed to be the name of a vertex or edge attribute to be extracted and transformed
d	is an edgelist matrix of edge values optionally used by some edge attribute functions
edgetouse	numeric vector giving set of edge ids to be used (in case some edges are not being shown) required by some attributes

Details

Given a network object, the name of graphic parameter argument to `plot.network` and value, it will if necessary transform the value, or extract it from the network, according to the description in `plot.network`. For some attributes, if the value is the name of a vertex or edge attribute, the appropriate values will be extracted from the network before transformation.

Value

returns a vector with length corresponding to the number of vertices or edges (depending on the paramter type) giving the appropriately prepared values for the parameter type. If the values or specified attribute can not be processed correctly, and Error may occur.

Author(s)

skyebend@uw.edu

See Also

See also `plot.network`

Examples

```
net<-network.initialize(3)
set.vertex.attribute(net,'color',c('red','green','blue'))
set.vertex.attribute(net,'charm',1:3)
# replicate a single colorname value
plotArgs.network(net,'vertex.col','purple')
# map the 'color' attribute to color
plotArgs.network(net,'vertex.col','color')
# similarly for a numeric attribute ...
plotArgs.network(net,'vertex.cex',12)
plotArgs.network(net,'vertex.cex','charm')
```

`prod.network` *Combine Networks by Edge Value Multiplication*

Description

Given a series of networks, `prod.network` attempts to form a new network by multiplication of edges. If a non-null `attrname` is given, the corresponding edge attribute is used to determine and store edge values.

Usage

```
## S3 method for class 'network'
prod(..., attrname = NULL, na.rm = FALSE)
```

Arguments

...	one or more network objects.
attrname	the name of an edge attribute to use when assessing edge values, if desired.
na.rm	logical; should edges with missing data be ignored?

Details

The network product method attempts to combine its arguments by edgewise multiplication (*not* composition) of their respective adjacency matrices; thus, this method is only applicable for networks whose adjacency coercion is well-behaved. Multiplication is effectively boolean unless attrname is specified, in which case this is used to assess edge values – net values of 0 will result in removal of the underlying edge.

Other network attributes in the return value are carried over from the first element in the list, so some persistence is possible (unlike the multiplication operator). Note that it is sometimes possible to "multiply" networks and raw adjacency matrices using this routine (if all dimensions are correct), but more exotic combinations may result in regrettably exciting behavior.

Value

A network object.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network.operators

Examples

```
#Create some networks
g<-network.initialize(5)
h<-network.initialize(5)
i<-network.initialize(5)
g[1:3,,names.eval="marsupial",add.edges=TRUE]<-1
h[1:2,,names.eval="marsupial",add.edges=TRUE]<-2
i[1,,names.eval="marsupial",add.edges=TRUE]<-3

#Combine by addition
pouch<-prod(g,h,i,attrname="marsupial")
pouch[,]                                  #Edge values in the pouch?
as.sociomatrix(pouch,attrname="marsupial")     #Recover the marsupial
```

read.paj *Read a Pajek Project or Network File and Convert to an R 'Network' Object*

Description

Return a (list of) network object(s) after reading a corresponding .net or .paj file. The code accepts ragged array edgelists, but cannot currently handle 2-mode, multirelational (e.g. KEDS), or networks with entries for both edges and arcs (e.g. GD-a99m). See network, statnet, or sna for more information.

Usage

```
read.paj(
  file,
  verbose = FALSE,
  debug = FALSE,
  edge.name = NULL,
  simplify = FALSE,
  time.format = c("pajekTiming", "networkDynamic")
)
```

Arguments

file	the name of the file whence the data are to be read. If it does not contain an absolute path, the file name is relative to the current working directory (as returned by getwd). file can also be a complete URL.
verbose	logical: Should longer descriptions of the reading and coercion process be printed out?
debug	logical: Should very detailed descriptions of the reading and coercion process be printed out? This is typically used to debug the reading of files that are corrupted on coercion.
edge.name	optional name for the edge variable read from the file. The default is to use the value in the project file if found.
simplify	Should the returned network be simplified as much as possible and saved? The values specifies the name of the file which the data are to be stored. If it does not contain an absolute path, the file name is relative to the current working directory (see getwd). If specify is TRUE the file name is the name file.
time.format	if the network has timing information attached to edges/vertices, how should it be processed? 'pajekTiming' will attach the timing information unchanged in an attribute named pajek.timing. 'networkDynamic' will translate it to a spell matrix format, attach it as an 'activity' attribute and add the class 'networkDynamic' – formating it for use by the networkDynamic package.

Details

If the *Vertices block includes the optional graphic attributes (coordinates, shape, size, etc.) they will be read attached to the network as vertex attributes but values will not be interperted (i.e. Pajek's color names will not be translated to R color names). Vertex attributes included in a *Vector block will be attached as vertex attributes.

Edges or Arc weights in the *Arcs or *Edges block are include in the network as an attribute with the same name as the network. If no weight is included, a default weight of 1 is used. Optional graphic attributes or labels will be attached as edge attributes.

If the file contains an empty Arcs block, an undirected network will be returned. Otherwise the network will be directed, with two edges (one in each direction) added for every row in the *Edges block.

If the *Vertices, *Arcs or *Edges blocks having timing information included in the rows (indicated by ... tokens), it will be attached to the vertices with behavior determined by the time.format option. If the 'networkDynamic' format is used, times will be translated to networkDynamic's spell model with the assumtion that the original Pajek representation was indicating discrete time chunks. For example "[5-10]" will become the spell [5,11], "[2-*]" will become [2,Inf] and "[7]" will become [7,8]. See documentation for networkDynamic's ?activity.attribute for details.

The *Arcslist, *Edgelist and *Events blocks are not yet supported.

As there is no known single complete specification for the file format, parsing behavior has been infered from references and examples below.

Value

The structure of the object returned by read.paj depends on the contents of the file it parses.

- if input file contains information about a single 'network' object (i.e .net input file) a single network object is returned with attribute data set appropriately if possible. or a list of networks (for .paj input).
- if input file contains multiple sets of relations for a single network, a list of network objects ('network.series') is returned, along with a formula object?.
- if input .paj file contains additional information (like partition information), or multiple *Network definitions a two element list is returned. The first element is a list of all the network objects created, and the second is a list of partitions, etc. (how are these matched up)

Author(s)

Dave Schruth <dschruth@u.washington.edu>, Mark S. Handcock <handcock@stat.washington.edu> (with additional input from Alex Montgomery <ahm@reed.edu>), Skye Bender-deMoll <skyebend@uw.edu>

References

Batagelj, Vladimir and Mrvar, Andrej (2011) Pajek Reference Manual version 2.05 http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf Section 5.3 pp 73-79

Batageli, Vladimir (2008) "Network Analysis Description of Networks" http://vlado.fmf.uni-lj.si/pub/networks/doc/ECPR/08/ECPR01.pdf

Pajek Datasets http://vlado.fmf.uni-lj.si/pub/networks/data/esna/

See Also

network

Examples

```
## Not run:
require(network)

par(mfrow=c(2,2))

test.net.1 <- read.paj("http://vlado.fmf.uni-lj.si/pub/networks/data/GD/gd98/A98.net")
plot(test.net.1,main=test.net.1%n%'title')

test.net.2 <- read.paj("http://vlado.fmf.uni-lj.si/pub/networks/data/mix/USAir97.net")
# plot using coordinates from the file in the file
plot(test.net.2,main=test.net.2%n%'title',
               coord=cbind(test.net.2%v%'x',
               test.net.2%v%'y'),
               jitter=FALSE)

# read .paj project file
# notice output has $networks and $partitions
read.paj('http://vlado.fmf.uni-lj.si/vlado/podstat/AO/net/Tina.paj')

## End(Not run)
```

---

sum.network *Combine Networks by Edge Value Addition*

Description

Given a series of networks, sum.network attempts to form a new network by accumulation of edges. If a non-null attrname is given, the corresponding edge attribute is used to determine and store edge values.

Usage

```
## S3 method for class 'network'
sum(..., attrname = NULL, na.rm = FALSE)
```

Arguments

...	one or more network objects.
attrname	the name of an edge attribute to use when assessing edge values, if desired.
na.rm	logical; should edges with missing data be ignored?

Details

The network summation method attempts to combine its arguments by addition of their respective adjacency matrices; thus, this method is only applicable for networks whose adjacency coercion is well-behaved. Addition is effectively boolean unless `attrname` is specified, in which case this is used to assess edge values – net values of 0 will result in removal of the underlying edge.

Other network attributes in the return value are carried over from the first element in the list, so some persistence is possible (unlike the addition operator). Note that it is sometimes possible to "add" networks and raw adjacency matrices using this routine (if all dimensions are correct), but more exotic combinations may result in regrettably exciting behavior.

Value

A network object.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

network.operators

Examples

```
#Create some networks
g<-network.initialize(5)
h<-network.initialize(5)
i<-network.initialize(5)
g[1,,names.eval="marsupial",add.edges=TRUE]<-1
h[1:2,,names.eval="marsupial",add.edges=TRUE]<-2
i[1:3,,names.eval="marsupial",add.edges=TRUE]<-3

#Combine by addition
pouch<-sum(g,h,i,attrname="marsupial")
pouch[,]                                    #Edge values in the pouch?
as.sociomatrix(pouch,attrname="marsupial")     #Recover the marsupial
```

`valid.eids` *Get the ids of all the edges that are valid in a network*

Description

Returns a vector of valid edge ids (corresponding to non-NULL edges) for a network that may have some deleted edges.

Usage

```
valid.eids(x, ...)

## S3 method for class 'network'
valid.eids(x, ...)
```

Arguments

x	a network object, possibly with some deleted edges.
...	additional arguments to methods.

Details

The edge ids used in the network package are positional indices on the internal "mel" list. When edges are removed using `delete.edges` `NULL` elements are left on the list. The function `valid.eids` returns the ids of all the valid (non-null) edge ids for its `network` argument.

Value

a vector of integer ids corresponding to the non-null edges in x

Note

If it is known that x has no deleted edges, `seq_along(x$mel)` is a faster way to generate the sequence of possible edge ids.

Author(s)

skyebend

See Also

See also `delete.edges`

Examples

```
net<-network.initialize(100)
add.edges(net,1:99,2:100)
delete.edges(net,eid=5:95)
# get the ids of the non-deleted edges
valid.eids(net)
```

`which.matrix.type` *Heuristic Determination of Matrix Types for Network Storage*

Description

`which.matrix.type` attempts to choose an appropriate matrix expression for a `network` object, or (if its argument is a matrix) attempts to determine whether the matrix is of type adjacency, incidence, or edgelist.

Usage

```
which.matrix.type(x)
```

Arguments

`x`	a matrix, or an object of class `network`

Details

The heuristics used to determine matrix types are fairly arbitrary, and should be avoided where possible. This function is intended to provide a modestly intelligent fallback option when explicit identification by the user is not possible.

Value

One of `"adjacency"`, `"incidence"`, or `"edgelist"`

Author(s)

David Hunter <dhunter@stat.psu.edu>

References

Butts, C. T. (2008). "network: a Package for Managing Relational Data in R." *Journal of Statistical Software*, 24(2). https://www.jstatsoft.org/v24/i02/

See Also

`as.matrix.network`, `as.network.matrix`

Examples

```
#Create an arbitrary adjacency matrix
m<-matrix(rbinom(25,1,0.5),5,5)
diag(m)<-0

#Can we guess the type?
which.matrix.type(m)

#Try the same thing with a network
g<-network(m)
which.matrix.type(g)
which.matrix.type(as.matrix.network(g,matrix.type="incidence"))
which.matrix.type(as.matrix.network(g,matrix.type="edgelist"))
```

Index