The network Package

August 1, 2007

Version  1.2

Date   July 31, 2007

Title   Classes for Relational Data

Author  Carter T. Butts <buttsc@uci.edu>, with help from David Hunter <dhunter@stat.psu.edu>

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

Depends  R (>= 2.0.0), utils

Suggests  sna

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 version 2 (June, 1991) or later

R topics documented:

add.edges  ................................................................. 2
add.vertices ............................................................ 3
as.matrix.network ...................................................... 4
as.network.matrix ...................................................... 6
as.sociomatrix .......................................................... 7
attribute.methods ....................................................... 9
deletion.methods ...................................................... 11
edgeset.constructors ............................................... 13
emon ................................................................. 14
flo ................................................................. 16
get.edges ............................................................. 17
get.neighborhood ..................................................... 18
is.adjacent .......................................................... 19
network-internal ...................................................... 21
network/operators ................................................... 21
network-package ...................................................... 23
network .............................................................. 25
### Description

Add one or more edges to an existing network object.

### Usage

```r
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 such lists
- **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
add.vertices

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).

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..

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

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

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
g[,3]<-1
g[,]
Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} \text{an object of class \texttt{network}}
\item \texttt{nv} \hspace{1cm} \text{the number of vertices to add}
\item \texttt{vattr} \hspace{1cm} \text{optionally, a list of attributes with one entry per new vertex}
\end{itemize}

Details

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 \texttt{network}.

Value

Invisibly, a pointer to the updated \texttt{network} object; \texttt{add.vertices} modifies its argument in place.

Author(s)

Carter T. Butts \{buttc@uci.edu\}

References


See Also

\texttt{network}, \texttt{get.vertex.attribute}, \texttt{set.vertex.attribute}

Examples

\begin{verbatim}
#Initialize a network object
g<-network.initialize(5)
g

#Add five more vertices
add.vertices(g,5)
g
\end{verbatim}

---

\texttt{as.matrix.network} \hspace{1cm} \textit{Coerce a Network Object to Matrix Form}

Description

The \texttt{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.
## S3 method for class 'network':
\texttt{as.matrix(x, matrix.type = NULL, attrname = NULL, \ldots)}

## S3 method for class 'adjacency':
\texttt{as.matrix.network(x, attrname=NULL, \ldots)}

## S3 method for class 'edgelist':
\texttt{as.matrix.network(x, attrname=NULL, \ldots)}

## S3 method for class 'incidence':
\texttt{as.matrix.network(x, attrname=NULL, \ldots)}

### Arguments

\textbf{x} \hspace{1cm} \text{an object of class network}

\textbf{matrix.type} \hspace{1cm} \text{one of "adjacency","incidence","edgelist", or NULL}

\textbf{attrname} \hspace{1cm} \text{optionally, the name of an edge attribute to use for edge values}

\textbf{\ldots} \hspace{1cm} \text{additional arguments.}

### Details

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

Note that adjacency matrices may also be obtained using the extraction operator. See the relevant man page for details.

### Value

An adjacency, incidence, or edgelist matrix

### Author(s)

Carter T. Butts \langle buttsc@uci.edu \rangle and David Hunter \langle dhunter@stat.psu.edu \rangle

### References


### See Also

\texttt{which.matrix.type}, \texttt{network}, \texttt{network.extraction}

### Examples

#Create a random network
m <- matrix(rbinom(25,1,0.5),5,5)
diag(m) <- 0
g <- network(m)
# Coerce to matrix form
as.matrix.network(g, matrix.type = "adjacency")
as.matrix.network(g, matrix.type = "incidence")
as.matrix.network(g, matrix.type = "edgelist")

# Can also use the extraction operator
g[,] # Get entire adjacency matrix
g[1:5, 6:10] # 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

```r
## 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

- `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?

- `...`  
  additional arguments
as.sociomatrix

Details

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

Value

An object of class \texttt{network}

Author(s)

Carter T. Butts \{buttsc@uci.edu\} and David Hunter \{dhunter@stat.psu.edu\}

References


See Also

\texttt{edgeset.constructors, network, which.matrix.type}

Examples

```r
#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")
```

---

\texttt{as.sociomatrix} \textit{Coerce One or More Networks to Sociomatrix Form}

Description

\texttt{as.sociomatrix} takes adjacency matrices, adjacency arrays, \texttt{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

\texttt{as.sociomatrix(x, attrname = NULL, simplify = TRUE, ...)}
as.sociomatrix

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`)?
- **...**
  - 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

Put references to the literature/web site here

See Also

`as.matrix.network`, `network`

Examples

```r
#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.sociomat
as.sociomatrix(g,simplify=TRUE)  #Returns as-is
```
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**

```r
delete.edge.attribute(x, attrname)
delete.network.attribute(x, attrname)
delete.vertex.attribute(x, attrname)

get.edge.attribute(el, attrname, unlist = TRUE)
get.edge.value(x, attrname, unlist = TRUE)
get.network.attribute(x, attrname, unlist = FALSE)
get.vertex.attribute(x, attrname, na.omit = FALSE, null.na = TRUE, unlist = TRUE)
network.vertex.names(x)

list.network.attributes(x)
list.edge.attributes(x)
list.vertex.attributes(x)

set.edge.attribute(x, attrname, value, e=1:length(x$mel))
set.edge.value(x, attrname, value, e=1:length(x$mel))
set.network.attribute(x, attrname, value)
set.vertex.attribute(x, attrname, value, v=1:network.size(x))
```

**Arguments**

- **el**  
  a list of edges (possibly `network$mel`).

- **x**  
  an object of class `network`.

- **attrname**  
  the name of the attribute to get or set.

- **unlist**  
  logical; should retrieved attributes be `unlisted` prior to being returned?

- **na.omit**  
  logical; should values from missing vertices/edges be removed?

- **null.na**  
  logical; should `NULL` values be replaced with `NA`?

- **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.
attribute.methods

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. `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. 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.

Note that some attribute assignment/extraction can be performed through the various extraction/replacement operators. 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.

Author(s)

Carter T. Butts 〈buttsc@uci.edu〉

References


See Also

network, as.network.matrix, as.sociomatrix, as.matrix.network, network.extraction

Examples

```r
#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])

# 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")

# 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% "myeval"
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).
deletion.methods

Usage

```r
delete.edges(x, eid)
delete.vertices(x, vid)
```

Arguments

- `x`: an object of class `network`.
- `eid`: a vector of edge IDs.
- `vid`: a vector of vertex IDs.

Details

Note that an edge’s ID number corresponds to its order within `x$me1`. To determine edge IDs, see `get.edgeIDs`. Likewise, vertex ID numbers reflect the order with which vertices are listed internally (e.g., the order of `x$soel` 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.

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


See Also

- `get.edgeIDs`
- `network.extraction`
- `as.matrix.network`

Examples

```r
#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)
```
edgeset.constructors

Edgeset Constructors for Network Objects

Description

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

Usage

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, ...)  
network.bipartite(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 values?  
names.eval the edge attribute under which to store edge values, if any  
...     additional arguments to add.edge

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; code.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. Any additional command line parameters are passed to add.edge.

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.
Note
Handling of missing data is not yet fully implemented.

Author(s)
Carter T. Butts (buttsc@uci.edu) and David Hunter (dhunter@stat.psu.edu)

References

See Also
network, network.initialize, add.edge, 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

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. Hurricane Frederic network Hurricane Frederic SAR EMON
3. Lake Pomona network Lake Pomona SAR EMON
4. Mt. Si network Mt. Si SAR EMON
5. Mt. St. Helens network Mt. St. Helens SAR EMON
6. Texas Hill Country network Texas Hill Country SAR EMON
7. Wichita Falls 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command.Rank.Score</td>
<td>numeric</td>
<td>Mean rank in the command structure</td>
</tr>
<tr>
<td>Decision.Rank.Score</td>
<td>numeric</td>
<td>Mean rank in the decision process</td>
</tr>
<tr>
<td>Formalization</td>
<td>numeric</td>
<td>Degree of formalization</td>
</tr>
<tr>
<td>Location</td>
<td>character</td>
<td>Location code</td>
</tr>
<tr>
<td>Paid.Staff</td>
<td>numeric</td>
<td>Number of paid staff</td>
</tr>
<tr>
<td>Sponsorship</td>
<td>character</td>
<td>Sponsorship type</td>
</tr>
<tr>
<td>vertex.names</td>
<td>character</td>
<td>Organization name</td>
</tr>
<tr>
<td>Volunteer.Staff</td>
<td>numeric</td>
<td>Number of volunteer staff</td>
</tr>
</tbody>
</table>

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. 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**


**See Also**

`network`

**Examples**

```r
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**

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.edges(x, v, alter = NULL, neighborhood = c("out", "in", "combined"), na.omit = TRUE)
get.edgeIDs(x, v, alter=NULL, 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)
na.omit logical; should we omit missing edges?

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.) 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.

References


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.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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>an object of class network</td>
</tr>
<tr>
<td>( v )</td>
<td>a vertex ID</td>
</tr>
<tr>
<td>type</td>
<td>the neighborhood to be computed</td>
</tr>
<tr>
<td>na.omit</td>
<td>logical; should missing edges be ignored when obtaining vertex neighborhoods?</td>
</tr>
</tbody>
</table>
is.adjacent

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 〈butts@uci.edu〉

References


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")

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 = TRUE)
Arguments

- **x**: an object of class `network`
- **vi**: a vertex ID
- **vj**: a second vertex ID
- **na.omit**: logical; should missing edges count 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.)

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

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

References


See Also

- `get.neighborhood`, `network.extraction`

Examples

```r
# 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

[1,2]==1    # TRUE
[2,1]==1    # FALSE
```
Internal Network Package Functions

Description

Internal network functions.

Usage

is.discrete(x)

is.discrete.character(x)

is.discrete.numeric(x)

Arguments

x

an object, to be designated either discrete or continuous.

Details

Most of these are not to be called by the user.

See Also

network

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 %c% e2

! e1

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

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

\texttt{e1} \quad \text{an object of class network.}
\texttt{e2} \quad \text{another network.}

Details

A more useful “usage” statement than the above (but prohibited by R’s overzealous documentation restrictions from appearing there) is as follows:

\begin{align*}
  x + y \\
  x - y \\
  x * y \\
  x \%c\% y \\
  !x \\
  x | y \\
  x \& y
\end{align*}

In general, the binary network operators currently function by converting their arguments to adjacency matrices, carrying out the specified operation on those matrices, and then returning the result in network form. This is rather inefficient, and results in the loss of other attributes; such poor behavior is not guaranteed to be maintained in future versions of the network package.

Apart from the above, the specific operations carried out by these operators are mostly self-explanatory. One exception to this is \texttt{x \%c\% y}, which returns the network formed from the composition of graphs \texttt{x} and \texttt{y} (respectively). (Note that this may contain loops, whether or not the original networks allowed them.)

Slightly different behavior is exhibited by the unary operator, \texttt{!}, which returns the complement of its argument. The graph which is returned contains all attributes of the original, save for edge attributes (as none of the original edges are retained). Note that the complement of a large, sparse graph has \textit{many} edges, with concomitant memory consumption. When working with such graphs, consider whether some other mechanism (e.g., adjacency checking on the original graph) might prove more efficient.

Value

The resulting graph object.

Note

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

Author(s)

Carter T. Butts (buttsc@uci.edu)
network-package

References


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

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: 1.2
Date: July 31, 2007
Depends: R (>= 2.0.0), utils
Suggests: sna
License: GPL version 2 (June, 1991) or later

Index:

- add.edges: Add Edges to a Network Object
- add.vertices: Add Vertices to an Existing Network
- as.matrix.network: Coerce a Network Object to Matrix Form
- as.network.matrix: Coercion from Matrices to Network Objects
- as.sociomatrix: Coerce One or More Networks to Sociomatrix Form
- attribute.methods: Attribute Interface Methods for the Network Class
- deletion.methods: Remove Elements from a Network Object
- edgset.constructors: Edgeset Constructors for Network Objects
- emon: Interorganizational Search and Rescue Networks (Drabek et al.)
- flo: Florentine Wedding Data (Padgett)
- get.edges: Retrieve Edges or Edge IDs Associated with a Given Vertex
- get.neighborhood: Obtain the Neighborhood of a Given Vertex
- is.adjacent: Determine Whether Two Vertices Are Adjacent
- network: Networks
- network.arrow: Add Arrows or Segments to a Plot
- network.density: Compute the Density of a Network
- network.dyadcount: Return the Number of Edges in a Network Object
- network.edgecount: Return the Number of Edges in a Network Object
- 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
- plot.network.default: Two-Dimensional Visualization for Network Objects
network

Author(s)

Carter T. Butts <butts@uci.edu>, with help from Mark S. Handcock <handcock@stat.washington.edu>, David Hunter <dhunter@stat.psu.edu> and Martina Morris <morris@u.washington.edu>

Maintainer: Carter T. Butts <butts@uci.edu>

Description

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

Usage

network(x, vertex.attr=NULL, vertex.attrnames=NULL, directed=TRUE, hyper=FALSE, loops=FALSE, multiple=FALSE, bipartite = FALSE, ...)
network.copy(x)
x[[i]] <- value
as.network(x, ...)
is.network(x)
## S3 method for class 'network':
print(x, matrix.type=which.matrix.type(x),
     mixingmatrices=FALSE, na.omit=TRUE, ...)
## S3 method for class 'network':
summary(object, na.omit=TRUE, mixingmatrices=FALSE, ...)

Arguments

x for network, a matrix giving the network structure in adjacency, incidence, or edgelist form; otherwise, an object of class network.
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?
network

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.

i   index specifying an element to replace. (May be tacit during direct assignment.)

value   a network object to be set/copied.

matrix.type   one of "adjacency", "edgelist", "incidence".

object   an object of class network.

na.omit   logical; omit summarization of missing attributes in network?

mixingmatrices   logical; print the mixing matrices for the discrete attributes?

...   additional arguments.

Details

network constructs a network class object from a matrix representation.

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 network function if necessary.

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.

Author(s)

Carter T. Butts (buttsc@uci.edu) and David Hunter (dhunter@stat.psu.edu)

References

See Also

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

Examples

```r
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)
```

---

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**

```r
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`: Curve for arrow (can be a vector)
- `edge.steps`: Number of steps for drawing arrow (can be a vector)
network.density

Description

network.density computes the density of its argument.

Usage

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

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.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

- `network.edgecount`
- `network.size`

Examples

```r
# 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
```

Description

`network.dyadcount` returns the number of dyads within a network, removing those flagged as missing if desired.
network.dyadcount

Usage

network.dyadcount(x, na.omit = TRUE)

Arguments

  x                      an object of class network
  na.omit               logical; omit edges with na==TRUE from the count?

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).

Value

  The number of dyads in the network

Author(s)

  Mark S. Handcock (handcock@stat.washington.edu)

References

  Butts, C.T. 2002. “Memory Structures for Relational Data in R: Classes and Interfaces” Working
  Paper.

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.dyadcount(g)==6          # Verify the dyad count
  g<-network(m|t(m),directed=FALSE)
  network.dyadcount(g)==3          # nC2 in undirected case
network.edgecount  \hspace{1cm} \textit{Return the Number of Edges in a Network Object}

\section*{Description}

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

\section*{Usage}

\texttt{network.edgecount(x, na.omit = TRUE)}

\section*{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} an object of class \texttt{network}
  \item \texttt{na.omit} \hspace{1cm} logical; omit edges with \texttt{na==TRUE} from the count?
\end{itemize}

\section*{Details}

The return value \texttt{network.edgecount} is equal to \texttt{x\%n\%"mnext"-1}, minus the number of NULL edges (and missing edges, if \texttt{na.omit==TRUE}). Note that \texttt{g\%n\%"mnext"-1} cannot, by itself, be counted upon to be an accurate count of the number of edges!

\section*{Value}

The number of edges

\section*{Author(s)}

Carter T. Butts (buttsc@uci.edu)

\section*{References}


\section*{See Also}

\texttt{get.network.attribute}

\section*{Examples}

\begin{verbatim}
#Create a network with three edges
m<-matrix(0,3,3)
m[1,2]<-1; m[2,3]<-1; m[3,1]<-
gh<-network(m)
network.edgecount(g)==3  #Verify the edgecount
\end{verbatim}
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

```r
## S3 method for class 'network':
x[i, j, na.omit = TRUE]
## 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?
- **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).

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=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, value).

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)

See Also

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

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
```r
g[, names.eval="boo"]<-5
as.sociomatrix(g, "boo")
g %e% "hoo" <- "wah"
g %e% "hoo"
```

# Set/retrieve network and vertex attributes
```r
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

```r
has.loops(x)
is.bipartite(x)
is.directed(x)
is.hyper(x)
is.multiplex(x)
```

### Arguments

- `x` an object of class `network`

### 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 as 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. (Note that `is.bipartite` refers only to the storage properties of `x`; `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 ⟨butsc@uci.edu⟩

References


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 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.
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


See Also

`network`, `as.network.matrix`

Examples

```r
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

```r
network.layout.circle(d, layout.par)  
network.layout.fruchtermanreingold(d, layout.par)  
network.layout.kamadakawai(d, layout.par)
```

Arguments

- **d**: an adjacency matrix, 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, `d`, must be the (dichotomous) graph adjacency matrix;
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, NROW(d))`, whose rows contain the vertex coordinates.

Other than this, anything goes. (In particular, note that `layout.par` could be used to pass additional matrices, if needed.)

The `network.layout` functions currently supplied by default are as follows:

- **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$niter`  This argument controls the number of iterations to be employed. (Defaults to 500.)
  - `layout.par$max.delta`  Sets the maximum change in position for any given iteration. (Defaults to `NROW(d)`.)
  - `layout.par$area`  Sets the “area” parameter for the F-R algorithm. (Defaults to `NROW(d)^2`.)
  - `layout.par$cool.exp`  Sets the cooling exponent for the annealer. (Defaults to 3.)
  - `layout.par$repulse.rad`  Determines the radius at which vertex-vertex repulsion cancels out attraction of adjacent vertices. (Defaults to `area*NROW(d)`.)
  - `layout.par$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 `NROW(d)/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$kkconst`  Sets the Kamada-Kawai vertex attraction constant. (Defaults to `NROW(d)^2`.)
  - `layout.par$elen`  Provides the matrix of interpoint distances to be approximated. (Defaults to the geodesic distances of `d` after symmetrizing, capped at `sqrt(NROW(d))`.)
  - `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 ⟨butts@uci.edu⟩

References


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, ...)
network.loop

  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 {butts@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.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

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


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

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

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).
- `...`: 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)

See Also

`plot.network`, `polygon`
Examples

```r
# 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

```r
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 `{buttc@uci.edu}`

References

plot.network.default

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 = FALSE, boxed.labels = TRUE, label.pos = 0,
label.bg = "white", vertex.sides = 8, vertex.rot = 0,
arrowshead.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, 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`.
- **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`.
- **coord**: user-specified vertex coordinates, in an NCOL(dat)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.
- **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.
- **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.

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.

...  additional arguments to plot.
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 (butts@uci.edu)

References


See Also

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

Examples

```r
#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),
```
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)

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). Also, this archive currently only contains networks under 10,000 nodes. See network, statnet, or sna for more information.

Usage

read.paj(file, verbose = FALSE, debug = FALSE, edge.name = NULL, simplify = FALSE)

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.

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.
Value

read.paj returns a 'network' object (for .net input) or a list of networks (for .paj input). Additional information in the .paj file (like partition information) is attached to the network objects in another higher order list.

Author(s)

Dave Schruth (dschruth@u.washington.edu), Mark S. Handcock (handcock@stat.washington.edu)

See Also

statnet, network, networkdata

Examples

```r
## 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$gal$title)
test.net.2 <- read.paj("http://vlado.fmf.uni-lj.si/pub/networks/data/mix/USAir97.net")
plot(test.net.2,main=test.net.2$gal$title)
## 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

```r
## 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 \texttt{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 \texttt{network} object.

Author(s)

Carter T. Butts ⟨butsc@uci.edu⟩

See Also

\texttt{network.operators}

Examples

```r
#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
```

Description

\texttt{which.matrix.type} \hspace{1em} \textit{Heuristic Determination of Matrix Types for Network Storage}

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

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)

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

!.network (network.operators), 21
+Topic aplot
   network.arrow, 27
   network.loop, 38
   network.vertex, 40
+Topic arith
   prod.network, 46
   sum.network, 48
+Topic classes
   add.edges, 1
   add.vertices, 3
   as.matrix.network, 4
   as.network.matrix, 5
   attribute.methods, 8
   deletion.methods, 11
   edgeset.constructors, 12
   get.edges, 16
   network, 24
   network.dyadcount, 29
   network.edgecount, 30
   network.indicators, 33
   network.initialize, 35
   network.size, 39
+Topic datasets
   emon, 14
   flo, 16
   read.paj, 47
+Topic dplot
   network.layout, 36
+Topic graphs
   add.edges, 1
   add.vertices, 3
   as.matrix.network, 4
   as.network.matrix, 5
   as.sociomatrix, 7
   attribute.methods, 8
   deletion.methods, 11
   edgeset.constructors, 12
   get.edges, 16
   get.neighborhood, 18
   is.adjacent, 19
   network, 24
   network.arrow, 27
   network.density, 28
   network.dyadcount, 29
   network.edgecount, 30
   network.indicators, 33
   network.initialize, 35
   network.layout, 36
   network.loop, 38
   network.operators, 21
   network.size, 39
   network.vertex, 40
   permute.vertexIDs, 41
   plot.network.default, 42
   prod.network, 46
   sum.network, 48
   which.matrix.type, 50
+Topic hplot
   plot.network.default, 42
+Topic internal
   network-internal, 20
+Topic manip
   as.sociomatrix, 7
   network.extraction, 31
   permute.vertexIDs, 41
+Topic math
   network.operators, 21
+Topic package
   network-package, 23
   *(network-internal), 20
   *.network(network.operators), 21
   +(network-internal), 20
   +.network(network.operators), 21
   -(network-internal), 20
   -.network(network.operators), 21
   [.network(network.extraction), 31
INDEX

[<-.network (network.extraction), 31
$<-.network (network), 24
%e% (network.operators), 21
%e% (network.extraction), 31
%eattr% (network.extraction), 31
%eattr% (network.extraction), 31
%n% (network.extraction), 31
%n% (network.extraction), 31
%nattr% (network.extraction), 31
%nattr% (network.extraction), 31
%v% (network.extraction), 31
%v% (network.extraction), 31
%vattr% (network.extraction), 31
%vattr% (network.extraction), 31
&.network (network.operators), 21
| .network (network.operators), 21
add.edge, 13
add.edge (add.edges), 1
add.edges, 1, 33, 34
add.vertices, 2, 3
arrows, 27
as.matrix.network, 4, 7, 8, 10, 26, 50
as.network (network), 24
as.network.default
(as.network.matrix), 5
as.network.matrix, 5, 10, 26, 35, 50
as.sociomatrix, 7, 10, 33
attribute.methods, 8, 26, 33
delete.edge.attribute
(attribute.methods), 8
delete.edges, 2
delete.edges (deletion.methods), 11
delete.network.attribute
(attribute.methods), 8
delete.vertex.attribute
(attribute.methods), 8
delete.vertices, 41
delete.vertices
(deletion.methods), 11
deletion.methods, 11, 26
edgeset.constructors, 6, 12, 26
emon, 14
flo, 16
get.edge.attribute
(attribute.methods), 8
glot.edge.value
(attribute.methods), 8
glot.edgeIDs, 11, 12
glot.edgeIDs (get.edges), 16
glot.edges, 16, 18
glot.neighborhood, 17, 18, 20
glot.network.attribute, 30, 31, 34, 40
glot.network.attribute
(attribute.methods), 8
glot.vertex.attribute, 3
glot.vertex.attribute
(attribute.methods), 8
glotwd, 47, 48
gplot, 45
gplot.arrow, 28
gplot.layout, 37
gplot.loop, 39
gplot.vertex, 41
has.loops (network.indicators), 33
is.adjacent, 18, 19, 33
is.bipartite
(network.indicators), 33
is.directed (network.indicators), 33
is.discrete (network-internal), 20
is.hyper (network.indicators), 33
is.multiplex
(network.indicators), 33
is.network (network), 24
list.edge.attributes
(attribute.methods), 8
list.network.attributes
(attribute.methods), 8
list.vertex.attributes
(attribute.methods), 8
network, 2, 3, 5–8, 10, 13–16, 24, 34, 35, 41, 42, 45, 47–49
network-internal, 20
network-package, 23
network.adjacency
(edgeset.constructors), 12
set.vertex.attribute
  (attribute.methods), 8
sna, 22
statnet, 48
sum.network, 48
summary.network (network), 24
switchArcDirection (read.paj), 47
unlist, 9
which.matrix.type, 4–6, 50

permute.vertexIDs, 41
plot, 45
plot.network, 26–28, 36–41
plot.network
  (plot.network.default), 42
plot.network.default, 42
polygon, 27, 28, 38–41
print.network (network), 24
print.summary.network (network), 24
prod.network, 46
read.paj, 47
readAndVectorizeLine (read.paj), 47

segments, 27
set.edge.attribute
  (attribute.methods), 8
set.edge.value
  (attribute.methods), 8
set.network.attribute
  (attribute.methods), 8
set.vertex.attribute, 3

network.arrow, 27, 38, 39, 45
network.bipartite
  (edgeset.constructors), 12
network.density, 28
network.dyadcount, 29
network.edgecount, 29, 30
network.edgelist
  (edgeset.constructors), 12
network.extraction, 2, 5, 10, 12, 13, 20, 22, 31
network.incidence
  (edgeset.constructors), 12
network.indicators, 2, 26, 33
network.initialize, 13, 26, 35
network.layout, 36, 43, 45
network.loop, 28, 38, 45
network.operators, 21, 33, 47, 49
network.size, 29, 39
network.vertex, 40, 45
network.vertex.names
  (attribute.methods), 8
network.vertex.names
  (attribute.methods), 8
networkdata, 48

unlist, 9

which.matrix.type, 4–6, 50