Package 'timeordered'

August 21, 2023

Type Package

Title Time-Ordered and Time-Aggregated Network Analyses

Version 1.0.0

Date 2023-08-19

Author Benjamin Blonder

Maintainer Benjamin Blonder <benjamin.blonder@berkeley.edu>

Description Approaches for incorporating time into network analysis. Methods include: construction of time-ordered networks (temporal graphs); shortest-time and shortest-path-length analyses; resource spread calculations; data resampling and rarefaction for null model construction; reduction to time-aggregated networks with variable window sizes; application of common descriptive statistics to these networks; vector clock latencies; and plotting functionalities. The package supports <doi:10.1371/journal.pone.0020298>.

License GPL-3

LazyLoad yes

LazyData true

Depends igraph, plyr

Enhances

BuildManual no

Imports methods

NeedsCompilation no

Repository CRAN

Date/Publication 2023-08-20 22:20:02 UTC

R topics documented:

`ants` *Ant interaction data*

Description

From a recent study of information flow in ant colonies. In this study, ants were uniquely marked with paint and identified by a four letter code - e.g. WGWB denotes an ant with a red head, green thorax, white left gaster, and blue right gaster. Body positions with missing paint marks are denoted with underscores.

In-nest activity was recorded with a high definition video camera. The complete set of pairwise interactions between all individuals at all times was obtained by several undergraduates repeatedly watching each video. Interactions were defined as any touch between one ant's antenna and any body part of another ant.

The dataset contains four columns: VertexFrom, VertexTo, TimeStart, and TimeStop. Each row is a unique interaction between two ants. Each interaction is directed, indicating that the VertexFrom ant has initiated a contact with the VertexTo ant. TimeStart and TimeStop characterize when the interaction began and finished. In this demo version of the data set, TimeStop = TimeStart + 1. Times are recorded in seconds.

Usage

```
ants
```

Format

A data frame containing 1911 observations over 24 minutes.

Source

Blonder & Dornhaus (2011), Supplementary Information, Colony 1-1.

References

Blonder & Dornhaus, *Time-ordered networks reveal limitations to information flow in ant colonies*. PLoS One (2011), in press.

`applynetworkfunction` *Applies a function (typically a descriptive statistic) to multiple time-aggregated networks*

Description

-

Usage

```
applynetworkfunction(slices, fun)
```

Arguments

`slices`	A list of time-aggregated networks, of class igraph
`fun`	The function to be applied; takes a single argument

Value

A list whose entries represent the function's value for each network

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`generatenetworkslices`,

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
td100 <- generatetimedeltas(0,1500,100)
ns100 <- generatenetworkslices(g, td100)
md100 <- applynetworkfunction(ns100, diameter)
tl100 <- generatetimelags(0,1500,100)
nl100 <- generatenetworkslices(g, tl100)
ml100 <- applynetworkfunction(nl100, function(x){diameter(x)})
par(mfrow=c(1,2))
plot(midpoints(td100),unlist(md100),type="l",xlab="Time (window size = 100)",ylab="Diameter")
plot(maxpoints(tl100),unlist(ml100),type="l",xlab="Aggregation time",ylab="Diameter")
```

---

generatelatencies *Generates vector-clock latencies for each individual at each time.*

Description

Vector clock latencies describe the minimum time delay between one individual broadcasting a signal and another individual receiving it, at a given time, through any causally permitted path in the time-ordered network. Smaller values indicate individuals that are connected by shorter causally-permitted paths at a given time.

Usage

```
generatelatencies(raw, allindivs)
```

Arguments

`raw`	An event list, consisting of a data frame with four columns: VertexFrom, VertexTo, TimeStart, and TimeStop. Each row in this data frame represents a single directed interaction event between VertexFrom and VertexTo beginning at TimeStart and ending at TimeStop. Assumes that no event begins at a time less than zero.
`allindivs`	A list of all possible vertices including ones not observed interacting during the range of time reported in `raw`.

Value

A n x n x m array, where n is the number of vertices and m is the maximum start time in the raw event list. The [i,j,k] entry of the array describes the latency from i to j at time k. NA is returned if there is not causally permitted path between i and j by time k.

Note

Return value can require large memory allocation depending on the data set. Ensure that data contains no times < 0 before running.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

References

Kossinets et al. The structure of information pathways in a social communication network. KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (2008)

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
l <- generatelatencies(ants, allindivs)
image(l[,,1000],axes=FALSE,frame=TRUE,col=rainbow(100))
axis(1, at = (1:ncol(l))/ncol(l), labels=colnames(l),tick=FALSE,las=2,cex.axis=0.2)
axis(2, at = (1:nrow(l))/nrow(l), labels=rownames(l),tick=FALSE,las=2,cex.axis=0.2)
```

---

generatenetworkslices *Generates multiple time-aggregated networks from a time-ordered network*

Description

Constructs weighted directed networks from all events occurring within certain time windows. Weight is equal to the number of interactions observed during the time window.

Usage

```
generatenetworkslices(g, timedeltas)
```

Arguments

g	The time-ordered network to be sliced.
timedeltas	A n x 2 matrix, where each row contains a set of start (first column) and stop (second column) times at which the network should be sliced.

Value

A list containing n time-aggregated networks corresponding to the n time windows.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

plotnetworkslices, generatetimedeltas, generatetimelags~~~

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
td100 <- generatetimedeltas(0,1500,100)
ns100 <- generatenetworkslices(g, td100)
plotnetworkslices(ns100, td100)
```

`generatetimeaggregatednetwork`

Constructs a weighted time-aggregated network from a time-ordered network by aggregating interactions occurring between a start and stop time. Weights are stored as E(g)$weight.

Description

-

Usage

```
generatetimeaggregatednetwork(g, starttime, stoptime)
```

Arguments

`g`	The time-ordered network to be aggregated
`starttime`	The time at which to begin aggregating interactions.
`stoptime`	The time at which to stop aggregating interactions.

Value

A weighted time-aggregated network whose edge weights equal the number of interactions between those vertices in the time window.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`generatenetworkslices`

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
tan500 <- generatetimeaggregatednetwork(g, 0, 500)
plottanet(tan500)
```

`generatetimedeltas` *Constructs matrix of sequential time windows suitable for slicing time ordered networks*

Description

-

Usage

```
generatetimedeltas(starttime, stoptime, delta)
```

Arguments

`starttime`	The starting time of the first time window.
`stoptime`	The stopping time of the last time window.
`delta`	The size of each time window.

Value

A n x 2 matrix. Each row contains the start and stop time of a window with width delta.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`generatetimelags` ~~~

Examples

```
td100 <- generatetimedeltas(0,1500,100)
boxplot(t(td100))
```

`generatetimelags` *Constructs matrix of increasingly large time windows suitable for assessing how window size affects time aggregated networks*

Description

-

Usage

```
generatetimelags(starttime, stoptime, delta)
```

Arguments

`starttime`	The starting time of the first time window.
`stoptime`	The stopping time of the last time window.
`delta`	The size by which to increase each time window.

Value

A n x 2 matrix. Each row contains the start and stop time of a window with widths increasing by delta.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`generatetimedeltas`

Examples

```
tl100 <- generatetimelags(0,1500,100)
boxplot(t(tl100))
```

generatetonetwork *Generates a time-ordered network from an interaction list.*

Description

Constructs a directed network describing the causally permitted paths between a set of vertices that interact at known times.

Usage

```
generatetonetwork(raw, allindivs)
```

Arguments

raw	An event list, consisting of a data frame with four columns: VertexFrom, VertexTo, TimeStart, and TimeStop. Each row in this data frame represents a single directed interaction event between VertexFrom and VertexTo beginning at TimeStart and ending at TimeStop.
allindivs	A list of all possible vertices potentially including ones not observed interacting during the range of time reported in raw. Defaults to the vertices observed in raw.

Value

A weighted directed network of class 'igraph'. Each vertex represents an individual at a time during which an interaction occurred. Edges represent causally permitted paths of resource flow and have a TimeCost, describing the time between interactions for an individual, or is 0 if the edge represents an interaction, and a HopCost, which is 0 if the edge connects the same individual at multiple times and 1 if it connects different individuals at the same time.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

References

Kostakos V. Temporal Graphs. arXiv (2008) vol. physics.soc-ph

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
plottonet(g)
```

`generatetonetworkfromvel` *Generates a time-ordered network from a data frame listing all directed edges. An internal function.*

Description

-

Usage

```
generatetonetworkfromvel(vel)
```

Arguments

`vel`	A data frame listing all directed edges

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`generatevertexedgelist` *Generates a data frame listing all directed edges in a time-ordered network from an observed interaction list. An internal function.*

Description

-

Usage

```
generatevertexedgelist(raw, allindivs)
```

Arguments

`raw`	A data frame of events
`allindivs`	A vector of names

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

maxpoints *Determines the maximum value of each row of a matrix; used as a convenience function for plotting.*

Description

-

Usage

```
maxpoints(td)
```

Arguments

td	A n x 2 matrix describing a set of start and stop times.

Value

A maximum value for each of n rows of td

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

generatetimelags,generatetimedeltas ~~~

Examples

```
tl100 <- generatetimelags(0,1500,100)
boxplot(t(maxpoints(tl100)))
```

midpoints *Determines the mean value of each row of a matrix; used as a convenience function for plotting.*

Description

-

Usage

```
midpoints(td)
```

Arguments

td	A n x 2 matrix describing a set of start and stop times.

Value

A mean value for each of n rows of td

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

generatetimelags,generatetimedeltas ~~~

Examples

```
tl100 <- generatetimelags(0,1500,100)
boxplot(t(midpoints(tl100)))
```

---

plotnetworkslices *Plots a time-aggregated network*

Description

-

Usage

```
plotnetworkslices(slices, timedeltas, ...)
```

Arguments

slices	A list of n time-aggregated networks
timedeltas	A n x 2 matrix describing the start and stop times for each time-aggregated network
...	Other arguments to be passed to igraph's plotting functionality

Value

None; used for its side effect of producing a plot.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

plotnetworkslices, generatetimedeltas, generatetimelags~~~

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
td100 <- generatetimedeltas(0,1500,100)
ns100 <- generatenetworkslices(g, td100)
plotnetworkslices(ns100, td100)
```

`plottanet` *Plots a time-aggregated network.*

Description

Plots a time-aggregated network. See igraph.plotting for more details.

Usage

```
plottanet(timeaggregatednetwork, layout = layout.circle,
vertex.label = V(timeaggregatednetwork)$name, vertex.size = 0,
vertex.label.cex = 0.5, edge.arrow.size = 0.5,
edge.width = E(timeaggregatednetwork)$Count/5, ...)
```

Arguments

`timeaggregatednetwork`	The network to print, an object of the igraph class
`layout`	Graph layout function - see ?layout in igraph
`vertex.label`	Vertex labels. Defaults to the name of each vertex.
`vertex.size`	Size of each vertex.
`vertex.label.cex`	Label size factor.
`edge.arrow.size`	Arrow size.
`edge.width`	Arrow width, defaults to be proportional to edge weight.
`...`	Other arguments to be passed to `igraph`'s plotting functionality

Value

None; used for its side effect of producing a plot.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

generatetimeaggregatednetwork

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
tan <- generatetimeaggregatednetwork(g, 0, 500)
plottanet(tan,layout=layout.kamada.kawai)
```

---

plottonet *Plots a time-ordered network.*

Description

Plots a time-ordered network with vertices ordinated along the x-axis and time increasing along the y-axis. Interactions are drawn as horizontal lines; vertices are connected to themselves in time by vertical lines.

Usage

```
plottonet(g, path = NULL, edgecolor = "gray",
edgehighlightcolor = "red", vertex.size = 0.01,
edge.arrow.size = 0.1, edge.width = 0.2,
vertex.color = NA, vertex.label.cex = 0.1,
vertex.frame.color = NA, vertex.label.color = "black")
```

Arguments

g	The time-ordered network to plot
path	If supplied, a particular list of vertices comprising a causally-permitted path that will be highlighted in the final illustration.
edgecolor	The color of all edges in the graph.
edgehighlightcolor	The color of the vertx path to be highlighted.
vertex.size	Vertex size. See igraph.plotting for more details.
edge.arrow.size	Edge arrow size.
edge.width	Edge width.
vertex.color	Vertex color.
vertex.label.cex	Vertex label size factor.
vertex.frame.color	Vertex frame color.
vertex.label.color	Vertex label color.

Value

None; used for its side-effect of producing a plot.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

generatetonetwork

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
plottonet(g)
```

---

randomizeidentities *Resamples data based on vertex identity.*

Description

Produces a new event list from an existing event list with resampled vertex identities given certain constraints on randomization. Effectively re-orders pairs of From/To vertices between different times.

Usage

```
randomizeidentities(raw, withinvertexfrom, byvertexfrom, withreplacement)
```

Arguments

raw	A raw event list to be resampled. Contains four columns: VertexFrom, VertexTo, TimeStart, TimeStop
withinvertexfrom	If true, resamples within data subsets where VertexFrom is fixed; otherwise re-samples within all data.
byvertexfrom	If true, subsets of data for withinvertexfrom are obtained using VertexFrom; if false, using VertexTo.
withreplacement	Samples with or without replacement.

Value

An event list of the same size or smaller as raw. The returned event list will be smaller only if resampling produces events that connect a vertex to itself; these are removed.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

randomizetimes,rarefy

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
ri <- randomizeidentities(ants,withinvertexfrom=TRUE,byvertexfrom=TRUE,withreplacement=TRUE)
g <- generatetonetwork(ri, allindivs)
plottonet(g)
```

---

randomizetimes *Resamples data based on event time.*

Description

Produces a new event list from an existing event list with resampled event times given certain constraints on randomization. Effectively re-orders pairs of start/stop times between different vertices.

Usage

```
randomizetimes(raw, withinvertexfrom, byvertexfrom, withreplacement)
```

Arguments

raw	A raw event list to be resampled. Contains four columns: VertexFrom, VertexTo, TimeStart, TimeStop
withinvertexfrom	If true, resamples within data subsets where VertexFrom is fixed; otherwise resamples within all data.
byvertexfrom	If true, subsets of data for withinvertexfrom are obtained using VertexFrom; if false, using VertexTo.
withreplacement	Samples with or without replacement.

Value

An event list of the same size as raw with event times resampled. Resampling does not break the relationship between start and stop time; i.e. resampled events will have the same duration as original events.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

randomizeidentities,rarefy

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
rt <- randomizetimes(ants,withinvertexfrom=TRUE,byvertexfrom=TRUE,withreplacement=TRUE)
g <- generatetonetwork(rt, allindivs)
plottonet(g)
```

---

randomize_edges_helper

Does all the work for edge_randomization *and* randomized_edges. *An internal function.*

Description

NA

Usage

```
randomize_edges_helper(edges, randomize_vertices)
```

Arguments

edges	A data frame for an edge list
randomize_vertices	A binary variable

Author(s)

Tim Gernat <mail@timgernat.name>

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (edges, randomize_vertices)
{
    vertex_columns <- c("VertexFrom", "VertexTo")
    unique_edges <- unique(edges[, vertex_columns])
    unique_edge_count <- nrow(unique_edges)
    edge_map <- cbind(unique_edges, unique_edges[sample(unique_edge_count,
        unique_edge_count), ])
    new_vertex_columns <- c("NewVF", "NewVT")
    colnames(edge_map) <- c(vertex_columns, new_vertex_columns)
    if (randomize_vertices) {
        edge_map[, new_vertex_columns] <- sample(unlist(edge_map[,
            new_vertex_columns]), unique_edge_count * 2)
        repeat {
            invalid <- (edge_map$NewVF == edge_map$NewVT) | (duplicated(edge_map[,
                new_vertex_columns]))
            if (sum(invalid) == 0)
                break
            for (i in which(invalid)) edge_map <- swap(edge_map,
                i, sample(new_vertex_columns, 1), sample(unique_edge_count,
                  1), sample(new_vertex_columns, 1))
        }
    }
    original_colnames <- colnames(edges)
    attribute_columns <- original_colnames[!(original_colnames %in%
        vertex_columns)]
    edges <- merge(edges, edge_map)
    edges <- edges[, c(new_vertex_columns, attribute_columns)]
    colnames(edges)[1:length(new_vertex_columns)] <- vertex_columns
    return(edges)
  }
```

`randomly_permuted_times` *Randomize temporal networks*

Description

Take a data frame specifying the edges of a temporal network and create a randomized reference network which maintains certain properties of the original network and destroys others.

Usage

```
total_randomization(edges)
```

```
randomly_permuted_times(edges)
vertex_randomization(edges)
contact_randomization(edges)
time_reversal(edges)
randomly_permuted_times(edges)
random_times(edges)
randomized_contacts(edges)
edge_randomization(edges)
randomized_edges(edges)
```

Arguments

edges	A data.frame of contacts specifying a temporal network. The data.frame has four columns: VertexFrom, VertexTo, TimeStart, and TimeStop. Each row represents a single directed contact between VertexFrom and VertexTo, beginning at TimeStart and ending at TimeStop. TimeStart and TimeStop may not be smaller than 0.

Details

randomly_permuted_times permutes the start time of contacts and adjusts the end time to maintain contact duration.

vertex_randomization assigns vertices randomly and with equal probability to contacts.

contact_randomization randomly permutes vertices between contacts.

time_reversal reverses the temporal order of contacts while maintaining the temporal distance of contacts.

randomly_permuted_times randomly permutes the start time of contacts while maintaining contact duration.

random_times assigns to the start time of each contact a random time between min(edges$TimeStart) and max(edges$TimeStop), maintaining the duration of each contact.

randomized_contacts redistributes contacts randomly among edges.

edge_randomization randomly exchanges whole contact sequences between edges.

randomized_edges randomly rewires edges. When an edge gets rewired, the contact sequence associated with that edge follow the edge.

total_randomization assigns vertices randomly to contacts, assuming that all vertices are equally likely participate in a contact

Randomized reference networks returned by these functions contain no contacts with self.

Value

A data.frame with the same columns as the edges, specifying the contacts of the randomized reference network.

Author(s)

Tim Gernat <mail@timgernat.name>

References

Holme & Saramaki, Physics Reports 519 (2012), p. 116-118

Examples

```
# load a temporal network
require(timeordered)
data(ants)

# randomly permute contact start timestamps while preserving contact duration
r1 <- randomly_permuted_times(ants)

# randomly permute vertices between contacts and assign a random start
# timestamp to each contact while preserving contact duration
r2 <- contact_randomization(ants)
r2 <- random_times(r1)
```

`rarefy` *Simulates the effect of insufficient sampling by data rarefaction.*

Description

Randomly removes a fixed fraction of the event list.

Usage

```
rarefy(raw, fraction)
```

Arguments

`raw`	The event list to be rarefied.
`fraction`	A fraction (between 0 and 1) of the events to be randomly deleted.

Value

An event list with floor(nrow(raw) * fraction) events remaining.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

randomizeidentities,randomizetimes

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`shortesthoppath` *Determines a path (shortest by the least number of unique vertices) between two vertices at two times.*

Description

-

Usage

```
shortesthoppath(g, startvertexname, startvertextime, stopvertexname, stopvertextime)
```

Arguments

`g`	The time-ordered network on which to find paths.
`startvertexname`	The name of the start vertex.
`startvertextime`	The time of the start vertex. Must be a time at which an interaction has occurred involving this vertex.
`stopvertexname`	The name of the stop vertex.
`stopvertextime`	The time of the stop vertex. Must be a time at which an interaction has occurred involving this vertex.

Value

A vertex list containing all the events on the shortest-hop path between the start and stop vertices/times.

Note

Multiple shortest-hop paths may exist; returns only one of them.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`shortesttimepath`

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
shp <- shortesthoppath(g, "WBGG", 927, "GYGG", 1423)
plottonet(g, shp)
title(paste(length(unique(shp$Name)),"hops"))
```

`shortesttimepath` *Determines a path (shortest by the least time) between a vertex at a start time and another vertex at any later time.*

Description

-

Usage

```
shortesttimepath(g, startvertexname, startvertextime, stopvertexname)
```

Arguments

`g`	The time-ordered network on which to find paths.
`startvertexname`	The name of the start vertex.
`startvertextime`	The time of the start vertex. Must be a time at which an interaction has occurred involving this vertex.
`stopvertexname`	The name of the stop vertex.

Value

A vertex list containing all the events on the shortest-time path between the start vertex at the start time and the stop vertex at a later time.

Note

May generate warning messages - don't worry!

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`shortesthoppath`

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
stp <- shortesttimepath(g, "WBGG", 927, "Q")
plottonet(g, stp)
title(paste(diff(range(stp$Time)), "time elapsed"))
```

`spreadanalysis` *Simulates the perfect spread of a resource on a time-ordered network.*

Description

Determines the number of unique vertices that can be causally linked to an interaction event after a certain time delay. This function determines the fraction of unique vertices reached after a certain time from a random sample of interaction events.

Usage

```
spreadanalysis(g, timedelays, numsamples, normalizebyname=FALSE)
```

Arguments

`g`	The time-ordered network to be studied.
`timedelays`	A vector time delays at which to determine the fraction of vertices reached.
`numsamples`	The number of random events to sample (without replacement) as seeds for the spreading process.
`normalizebyname`	If true, divides the number of vertices reached by the number of unique vertex names; if false, by the number of time-ordered vertices.

Value

A data frame whose columns are named for each time delay and contains the fraction of total vertices reached by a spreading process beginning from the seed vertices by the time delay.

Note

Results can be aggregated by start vertex - see `transformspreadbyindividual`

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`transformspreadbyindividual`

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
sa <- spreadanalysis(g, seq(0,1000,by=50), 20)
boxplot(sa[,-1],xlab="Time delay",ylab="Fraction reached")
```

---

swap *Swaps two elements in a data frame. An internal function.*

Description

NA

Usage

```
swap(df, r1, c1, r2, c2)
```

Arguments

df	A dataframe
r1	The first row to swap
c1	The first column to swap
r2	The second row to swap
c2	The second column to swap

Author(s)

Tim Gernat <mail@timgernat.name>

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (df, r1, c1, r2, c2)
{
    tmp <- df[r1, c1]
    df[r1, c1] <- df[r2, c2]
    df[r2, c2] <- tmp
    return(df)
  }
```

`transformspreadbyindividual` *A helper function to assess differences in spreading potential by vertex.*

Description

Converts a data frame of spreading samples into a data frame that is grouped by vertex identity.

Usage

```
transformspreadbyindividual(sa)
```

Arguments

`sa`	A data frame returned by `spreadanalysis`

Value

A data frame whose columns are the identities of vertices and whose rows are the mean fraction of vertices reached by the seed vertex at each time delay, averaged over all samples beginning at this vertex.

Author(s)

Benjamin Blonder <bblonder@email.arizona.edu>.

See Also

`spreadanalysis`

Examples

```
data(ants)
allindivs <- c(union(ants$VertexFrom, ants$VertexTo), "NULL1", "NULL2")
g <- generatetonetwork(ants, allindivs)
sa <- spreadanalysis(g, seq(0,1000,by=50), 20)
b <- transformspreadbyindividual(sa)
plot(ts(b),plot.type="single",col=rainbow(ncol(b)),xlab="Time",ylab="Fraction reached")
legend("bottomright",colnames(b),lwd=1,col=rainbow(ncol(b)),bg="white")
```

Index