Package: tsna (via r-universe)

October 12, 2024

Version 0.3.5

Date 2021-10-31

Title Tools for Temporal Social Network Analysis

Type Package

Depends network (>= 1.13), networkDynamic (>= 0.9), R (>= 3.0)

Imports statnet.common

Suggests networkDynamicData, testthat, sna, knitr, rmarkdown, relevent, ergm (>= 3.10)

Description Temporal SNA tools for continuous- and discrete-time longitudinal networks having vertex, edge, and attribute dynamics stored in the 'networkDynamic' format. This work was supported by grant R01HD68395 from the National Institute of Health.

License GPL-3

LazyLoad yes

URL http://statnet.org/

BugReports https://github.com/statnet/tsna/issues

VignetteBuilder knitr

Repository https://statnet.r-universe.dev

RemoteUrl https://github.com/statnet/tsna

RemoteRef HEAD

RemoteSha 3d39713428f0edb5a4365d9a35842da35a7d2728

Contents

`as.network.tPath` *Create a network object from a* `tPath` *object – the results of a path search*

Description

Takes the results of a temporal path search (a `tPath`) as input and creates a new network object representing the path tree which can be plotted for debugging, etc.

Usage

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

Arguments

`x`	A `tPath` list with several elements, the output of a path search.
`...`	possible additional arguments

Details

Attributes of original network are not copied

Value

a directed `networkDynamic` object representing the path information from the input. Each edge has the onset time indicated by its distance in the path.

Note

this is probably not the final form of this function

Author(s)

skyebend

See Also

See also paths.

Examples

```
data(moodyContactSim)
v1path<-tPath(moodyContactSim,v=1)
v1tree<-as.network(v1path)
plot(v1tree,displaylabels=TRUE)
```

---

duration *Compute Observed Activity Durations or Event Counts for Edges or Vertices*

Description

Computes observed activity durations or event counts for edges or vertices, aggregating at the edges, spell, or dyads level.

Usage

```
edgeDuration(nd, mode = c("duration", "counts"), subject = c("edges", "spells", "dyads"),
         e = seq_along(nd$mel), start = NULL, end = NULL, active.default = TRUE)

vertexDuration(nd, mode = c("duration", "counts"), subject = c("vertices", "spells"),
     v = seq_len(network.size(nd)), active.default = TRUE)
```

Arguments

nd	networkDynamic object to be evaluated
mode	option determining if the 'duration' of the spells should be summed, or simply the 'counts' of the number of spells. The later option is useful when the network contains events of zero duration.
subject	option describing the unit of analysis that the durations or counts should be aggregated at. 'spells' considers each event independently. For tVertexDuration, 'vertices' sums all the spells corresponding to a single vertex id. For edgeDuration, 'edges' sums all of the spells corresponding to a single edge id and 'dyads' sums all of the spells coresponding to a single (directed) tail-head pair (this means spells from multiplex edges conecting the same vertices will be added together).
e	numeric vector of edge ids to consider.
v	numeric vector of vertex ids to consider.
start	optional numeric value to be used to censor onset times. (not yet supported for vertices, must use net.obs.period instead.)
end	optional numeric value to be used to censor terminus times. (not yet supported for vertices, must use net.obs.period instead.)
active.default	logial, should edges or vertices with no timing information be considered always active by default?

Details

The function sums the durations of edge or vertex events or,in order to provide a useful metric for networks having zero-duration events, simply counts them. It is implemented by wrapping a call to `as.data.frame.networkDynamic` or `get.vertex.activity`. In many cases the output of the function will be fed to another statistical summary function like `summary` or `hist`. The level of aggregation can be selected by setting the subject to either `spells`, `edges`, or `dyads`.

Note that the 'observed' durations may not match the 'true' (statistically estimated) durations for a network due to the censoring of edges/vertices.

Value

A vector of values corresponding to the selected subjects with the count or duration of events. If the network contains no edges/vertices, `numeric(0)` will be returned.

Note

This is an early implementation of the function and its name and arguments are subject to change

Author(s)

skyebend

See Also

See also `as.data.frame.networkDynamic`

Examples

```
# look at the distributions of edge durations for
# a real-world contact network
## Not run:
require(networkDynamicData)
data(hospital_contact)
summary(edgeDuration(hospital,subject='edges'))
summary(edgeDuration(hospital,mode='counts',subject='edges'))

## End(Not run)
# look at the vertex durations for a network were
# vertices are not present every day
require(networkDynamic)
data(windsurfers)
vertexDuration(windsurfers)
```

formation_and_dissolution

Counts or fractions of edge transitions in a networkDynamic object

Description

The functions `tEdgeFormation` and `tEdgeDissolution` evaluate a network object at multiple time points and return counts (or fractions) of the number of edges forming (edge onset at time point) and dissolving (edge terminus at time point). The counts are returned as numeric vector which is a time-series object (class `ts`).

Usage

```
tEdgeFormation(nd, start, end, time.interval = 1,
               result.type=c('count','fraction'), include.censored=FALSE)

tEdgeDissolution(nd, start, end, time.interval = 1,
               result.type=c('count','fraction'), include.censored=FALSE)
```

Arguments

`nd`	a `networkDynamic` object
`start`	optional numeric time value at which evaluation should start (default is first observed time)
`end`	optional numeric time value at which evaluation should end (default is last observed time)
`time.interval`	optional numeric value giving time interval between evaluations (default is 1)
`result.type`	either `'count'` indicating that results should be returned as counts of tie changes or `'fraction'` meaning that results should be returned as fraction of ties dissolving (for dissolution) or fraction of empty dyads forming ties (for formation). In the later case the number of dyads is corrected for network directedness, loops, and bipartite partition size.
`include.censored`	logical, should ties with truncated/censored onset or termination times be included in the respective formation or dissolution counts?

Details

Uses `as.data.frame.networkDynamic` internally. TODO: dyad formation rate is not yet corrected for vertex activity, uses the aggregate, not the momentary, network size.

When `result.type='fraction'`:

- formation returns the ratio of number of ties forming to the number of possible empty dyads that could have formed ties. So value of 1 would mean all empty dyads formed ties, value of 0 means no ties formed. In sparse networks, the numbers will tend to be very, very small.

- dissolution returns the ratio of the number of ties dissolving to the number preexisting ties that could have dissolved. So value of 1 means all ties disolved, 0 means no ties disolved.

When `include.censored=FALSE` spells of edges which onset outside of the query range will not be included in formation counts.

Value

For `tEdgeFormation` and `tEdgeDissolution`, a numeric vector of class `ts` giving the formation and dissolution counts (respectively) as a time-series. For `edgeFormationAt` and `edgeDissolutionsAt`, a single numeric value

Note

should add additional args to allow binning other than 'at' for working with non-discrete time, options to deal with how censored edges are calculated.

Author(s)

skyebend@uw.edu

Examples

```
library(networkDynamicData)
data(concurrencyComparisonNets)
# plot formation and dissolution counts time-series
plot(tEdgeFormation(base),col='green',
  main='edge formation and dissolution rates per timestep of base')
points(tEdgeDissolution(base),col='red',type='l')
## Not run:
# compute fraction of ties dissolving every 10 steps
tEdgeDissolution(base,time.interval = 10,result.type = 'fraction')
# compute fraction of empty dyads forming ties every 10 steps
tEdgeFormation(base,time.interval = 10,result.type = 'fraction')

## End(Not run)
```

---

moodyContactSim *Jim Moody's example dynamic contact simulation network*

Description

A `networkDynamic` object containing the output of a simulation of 1000 timestep simulation of a sex contact network with 16 vertices and 18 edges. Each edge has a single activity spell.

Usage

```
data(moodyContactSim)
```

Format

A networkDynamic object.

Details

The object has a net.obs.period attribute describing the observation model. This is a useful network for testing path-based algorithms because it is small enough to visually inspect.

Source

Figure 5 of James Moody (2008) "Static Representations of Dynamic Networks" Duke Population Research Institute On-line Working Paper Series. http://www.soc.duke.edu/~jmoody77/StatDyn_5.pdf

Examples

```
data(moodyContactSim)
# plot a view of network with edge and vertex labels
plot(moodyContactSim,
     displaylabels=TRUE,
     label.cex=0.8,
     label.pos=5,
     vertex.col='white',
     vertex.cex=2,
     edge.label=sapply(get.edge.activity(moodyContactSim),function(e){
       paste('(',e[,1],'-',e[,2],')',sep='')
     }),
     edge.label.col='blue',
     edge.label.cex=0.8
   )
## Not run:
# data object was created with
moodyContactSim<-network.initialize(16,directed=FALSE)
tel<-matrix(c(674,701,1,9,
              214,247,1,11,
              621,651,1,12,
              583,615,1,16,
              749,793,11,8,
              719,745,8,13,
              712,739,13,5,
              634,660,13,3,
              769,795,13,7,
              453,479,13,4,
              494,524,13,2,
              224,256,7,10,
              40,72,10,4,
              665,692,4,14,
              709,740,2,15,
              575,599,2,16,
              748,782,4,16,
              701,733,16,6),
            ncol=4,byrow=TRUE)
```

```
moodyContactSim<-networkDynamic(moodyContactSim,edge.spells=tel)
obs<-moodyContactSim%n%'net.obs.period'
obs$mode<-'discrete'
obs$time.increment<-1
obs$time.unit<-'step'
obs$observations<-list(c(0,1000))
moodyContactSim%n%'net.obs.period'<-obs

## End(Not run)
```

---

paths *Temporally Reachable Paths in a networkDynamic Object*

Description

Functions to search out the sequence and distances of vertices in a networkDynamic object reachable from an initial vertex by following paths constrained by edge timing.

Usage

```
tPath(nd, v, direction=c('fwd','bkwd'),
                 type=c('earliest.arrive', 'latest.depart'),
                 start, end, active.default = TRUE, graph.step.time = 0)

is.tPath(x)
```

Arguments

nd	networkDynamic object to be searched for temporal paths
v	integer id of the vertex to be used as the starting point of the search
direction	option indicating the temporal direction in which the network should be searched: 'fwd' means search forwards in time and forward along edge directions, 'bkwd' means search backwards in time and backwards along edge directions.
type	option indicating the type of path (temporal constraint of the path) be searched for: • 'earliest.arrive' will find the paths that arrive first at the target vertices, • 'latest.depart' will find the paths that leave the source vertex at the latest possible time. Additional options will be added as implemented.
start	time at which to begin searching. Edges that terminate before this time will not be considered. If not specified, defaults to earliest time observed on the network according to get.change.times.
end	time to end the path search. Edges that onset on or after this time will not be considered in the path search.

`active.default` Boolean, default TRUE. Should edges with no timing information be considered active by default?

`graph.step.time`
numeric. How much time should be added for each edge traversal (graph hop)? Default is 0, meaning that path distances returned will be purely temporal and will not incorporate graph path distances and 'transmission' can cross multiple edges in a single instant. A value of 1 would correspond to counting path distances like a traditional centrality score or discrete time simulation.

`x` an object to be tested for inheriting the class `'tPath'`

Details

A *temporal path* in a dynamic network is a sequence of vertices and edges such that the onset times of successive elements are greater than or equal than those of the previous. In other words, the path is a directed traversal of the network that respects the constraints of edge activity spells and permits 'waiting' at intermediate vertices for 'future' edges to form.

When set to use `direction='fwd'`, `type='earliest.arrive'` `tPath` performs a time-minimizing Dijkstra's style Depth First Search to find the set of vertices reachable on a *forward temporal path* from the initial seed vertex `v` while respecting the constraints of edge timing.The path found is a *earliest arriving* (in contrast to the *earliest leaving* or *quickest* or *latest arriving* path). When there are multiple equivalent paths only a single one will be arbitrarily returned. NOTE THAT THE PATH-FINDING ALGORITHM WILL NOT GIVE CORRECT RESULTS IF ANY SPELLS CONTAIN VALUES LESS THAN 0.

When set to `direction='bkwd'` and `type='latest.depart'` the path will be found by searching backwards in time from the `end` point. In other words, it returns the set of vertices that can reach `v`, along with latest possible departure times from those vertices. Note that in this case the elapsed time values returned for `tdist` will be negative, indicating time measured backwards from the `end` bound.

When set to `type='fewest.steps'` the path returned will be a 'shortest' (fewest steps/graph hops) time-respecting path. This would not be necessiairly the quickest or earliest route, but would pass across the fewest possible number of edges (requires the fewest number of transmission steps).

The `graph.step.time` parmeter allows specifying an explicit duration for edge traversals. In this case the algorithm considers both the onset and terminus times of activity spells to ensure that suffecient time remains for an edge traversal to be made. If `graph.step.time` > the remaining duration of an edge's activity spell, the edge is considered non-traverseable. The primary use case for this parameter is to align the paths discovered with those that might be found by a discrete time transmission simulation in which a path can only spread a single graph hop per model timestep.

Vertex activity is currently ignored, and it is assumed that once a path reaches a vertex, all future edges from the vertex are accessible. The path search can be constrained in time using the `start` and `end` parameters to bound the time span to be explored by the path search.

`'bwkd'` `'latest.depart'` is essentially the inverse of fwd earliest arrive. It finds the latest time paths backwards from the initial seed vertex. This is the *latest-leaving* time. Note that the distance returned are positive, but represent the latest distance back in time from the `end` parameter time at which a vertex can reach `v`.

The `is.tPath` function checks if an object has the class `tPath`.

Value

Currently an object of class `tPath` which is essentially list with several elements providing information on the path found.

`tdist`	A numeric vector with length equal to network size in which each element contains the earliest/latest *temporal* distance at which the corresponding vertex could reach / be reached from the seed vertex. Values are elapsed time, as measured from the `start` parameter. Unreachable vertices are marked with `Inf`
`previous`	A numeric vector with length equal to network size in which each element indicates the previous vertex along (a possible) reachable path. Can be used to reconstruct the path tree. The initial vertex and unreachable vertices are marked with `0`
`gsteps`	A numeric vector (of length equal to network size) in which each element indicates the number of steps in the path (number of graph hops) to the vertex along the temporal path found starting at the seed vertex.
`start`	the numeric start value that was used as the earliest bound for the path calculation (may not have been explicitly set)
`end`	the numerid end value that was used as the latest bound for the path calculation (may not have been explicitly set)
`direction`	The direction `'fwd'` or `'bkwd'` of the path
`type`	The type of temporal constraint for the path

Note

Temporal distances are in terms of time measured from the `start` parameter, so to recover the model times at which each vertex was reached for forward paths use `$tdist+start` and backward paths with `end- $tdist`. This is an early draft of the function, its name and arguments are subject to change before release.

Author(s)

Skye Bender-deMoll

References

Unpublished discussions with James Moody and Martina Morris and the statnet team.

Useful background information (for a slightly different algorithm) can be found in: B. Bui Xuan, Afonso Ferreira, Aubin Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks." RR-4589, 2002. https://hal.inria.fr/inria-00071996/document

B. Bui Xuan, Afonso Ferreira, Aubin Jarry. Evolving graphs and least cost journeys in dynamic networks. WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, Mar 2003, Sophia Antipolis, France. 10 p., 2003 https://hal.inria.fr/inria-00466676/document

Examples

```
require(networkDynamicData)
data(hospital_contact)
hosPath<-tPath(hospital,v=1)
```

plotPaths *Network Plots a Highlighting Temporal Path(s) .*

Description

Wrapper for plot.network with appropriate defaults to plot a highlighted path, or over-plot highlighted paths on a on top of a static aggregate network plot.

Usage

```
plotPaths(nd, paths,  path.col = rainbow(length(paths), alpha = 0.5),
           displaylabels = TRUE, coord=NULL, ...)

## S3 method for class 'tPath'
plot(x, edge.col = "red",
              edge.label.col = edge.col,
              edge.lwd = 10,
              edge.label.cex = 0.7,
              displaylabels = TRUE,
              displayisolates = FALSE,
              jitter = FALSE,
              vertex.lwd = (x$gsteps == 0) * 4 + 1,
              vertex.cex = (x$gsteps == 0) * 1.5,
              vertex.col = NA, ...)
```

Arguments

nd	a networkDynamic object to be plotted.
paths	a tPath object containing temporal path (presumably extracted from nd) to be plotted over the network, or a list of paths to be plotted together on the same network.
path.col	vector of valid colors (possibly transparent) to be used for each path. Default will created semi-transparent colors from the rainbow palette.
x	object (assumed to be tPath) to be plotted on top of a static aggregate network plot
...	additional arguments to be passed to plot.network and plot.tPath.
coord	optional numeric matrix of coordinates for positioning vertices. See plot.network
edge.col	color for drawing edges (paths). See plot.network
edge.label.col	color for edge labels. Default to same color as edges. See plot.network
edge.lwd	numeric expansion factor for edge line widths. See plot.network
edge.label.cex	numeric expansion factor for edge labels. See plot.network
displaylabels	logical, should vertex labels be included on the plot? See plot.network
displayisolates	logical, should isolated vertices be included in the plot? See plot.network

jitter	adds random noise to positions (disabled by default) See plot.network
vertex.lwd	Vertex border line with. See plot.network
vertex.cex	Vertex expansion factor. Default is to scale up the origin vertex for the path, and not draw the other vertices. See plot.network
vertex.col	Color for vertices. Default is to leave them un-colored. See plot.network

Details

plotPaths plots the networkDynamic object using the normal plot.network function and ... arguments. Then calls plot.tPath for each tPath object in paths to over-plot the edges of path onto the network plot using the corresponding path.col color. Use of semi-transparent colors can help (somewhat) improve readability when paths overlap on the same edges.

plot.tPath plots the path information encoded in a single tPath object. It first creates a network using as.network.tPath and then calls plot.network with suitable defaults for drawing (or over-drawing) the path (doesn't display isolated vertices, draws times as edge labels, draws a color around the source vertex, etc.)

Value

Generates a network plot with a highlighted path, invisibly returns the plot coordinates.

Author(s)

skyebend

See Also

See also tPath

Examples

```
data(moodyContactSim)
v10path<-tPath(moodyContactSim,v=10,start=0)
# plot just the path from v10
plot(v10path)

# plot the path from v10 on top of the network
plotPaths(moodyContactSim,v10path)

# plot the paths from both v10 and v1
v1path<-tPath(moodyContactSim,v=1,start=0)
plotPaths(moodyContactSim,list(v10path,v1path))

# if ndtv package is installed, along with Graphviz system library,
# nice hierarchical trees can be drawn
## Not run:
plot(v10path,
     coord=network.layout.animate.Graphviz(
                  as.network(v10path),
                  layout.par = list(gv.engine='dot')
```

```
            ),
            jitter=FALSE
    )

## End(Not run)
```

---

pShiftCount *Compute Counts of Gibson's Participation Shifts*

Description

Uses the relevent package to compute counts of dyadic turn-taking events using a typology outlined by Gibson (2003)

Usage

```
pShiftCount(nd, start = NULL, end = NULL, output = c("final", "full"))
```

Arguments

nd	networkDynamic object to be evaluated
start	numeric initial time point to start evaluation from
end	numeric ending time point to finish evaluation
output	chracter value indicating if only the 'final' counts should be reported, or the 'full' matrix with updated counts for each event.

Details

Uses the accum.ps function in the package relevent to build counts of accumulated dyad participation shifts (turn-taking changes) using the dynamic information on tie changes represented in a directed networkDynamic object. The P-shifts are given in the order used in Gibson's 2003 Social Forces paper, namely:

- Turn Receiving:
  - [1] AB->BA (Alex talks to Brett, then Brett replies)
  - [2] AB->B0 (Alex talks to Brett, then Brett addresses the group)
  - [3] AB->BY (Alex talks to Brett, then Brett talks to Yuki)
- Turn Claiming:
  - [4] A0->X0 (Alex talks to the group, then Xuan talks to the group)
  - [5] A0->XA (Alex talks to the group, then Xuan talks to Alex)
  - [6] A0->XY (Alex talks to the group, then Xuan talks to Yuki)
- Turn Usurping:
  - [7] AB->X0 (Alex talks to Brett, then Xuan talks to the group)
  - [8] AB->XA (Alex talks to Brett, then Xuan talks to Alex)

  - [9] AB->XB (Alex talks to Brett, then Xuan talks to Brett)
  - [10] AB->XY (Alex talks to Brett, then Xuan talks to Yuki)
- Turn Continuing:

  [11] A0->AY (Alex talks to the group, then addresses Yuki)
  - [12] AB->A0 (Alex talks to Brett, then makes remark to the group)
  - [13] AB->AY (Alex talks to Brett, then to Yuki)

This uses Gibson's notation, in which A is the initial source, B is the initial target, X is a new (shifted) speaker, Y is a new (shifted) target, and 0 is used where no well-defined speaker or target is present. (Here, this would occurs when NA is given for source or destination, not currently supported)

It is worth noting that not all adjacent event pairs induce P-shifts, and hence the shift counts will not increment with every event. In particular, the first event does not induce a shift (since there is no prior event), and neither does a repetition of a previous event (e.g., AB->AB or A0->A0). The full set is thus affinely independent in general, although they will have a near (or even full) dimension of affine dependence on most data sets.

Event order is determined by sorting the network's edge spells by on the onset time of edge and then by the terminus. Gibson's typology assumes that edges/ties directed 'at the group' are distinguishable those directed at individuals, and has a strong assumption of sequential non-simultaneous events. Because the `networkDynamic` object does not explicitly clode for 'group' utterences, simultaneous edges originating from a speaker (same onset,terminus, and tail vertex) are assumed to be directed at the group, even if not all group members are reached by the ties.

Value

For `output='final'` (the default), the output is a matrix with one row containing counts for each of the 13 P-shift types accumulated over the time period requested. For `ouptut='full'`, the output is a data.frame with rows corresponding to each edge spell event. The first 13 rows are the counts of P-shift types, and the remaining four rows are the 'onset', 'terminus', 'tail', 'head', and a 'group' column indicating if the event was considered as a group-directed.

Author(s)

Carter Butts buttsc@uci.edu, Skye Bender-deMoll skyebend@uw.edu

References

Gibson, D.R. (2003) 'Participation Shifts: Order and Differentiation in Group Conversation' Social Forces 81 (4): 1335-1380 `https://doi.org/10.1353/sof.2003.0055`

Carter T. Butts (2008). A Relational Event Framework for Social Action. Sociological Methodology, 38(1), 155–200.

Examples

```
data(McFarland_cls33_10_16_96)
pShiftCount(cls33_10_16_96)
```

`reachable` *Find the set of vertices reachable from a given set using only paths moving forward in time*

Description

Does a breadth-first search from the specified set of vertices, respecting the direction and timing of edges. TODO: vertex activity.

Usage

```
forward.reachable(nd, v, start = NULL, end = NULL, per.step.depth = Inf)
```

Arguments

`nd`	a `network` (usually a `networkDynamic`) object
`v`	numeric vector giving the set of initial vertex.ids to start from
`start`	The beginning of the time range to start from
`end`	End of the time range to search to
`per.step.depth`	How many steps (default=1) to search per unit of time.

Details

The default value of `per.step.depth=Inf` is equivilent to assuming that the 'process' takes no time to travel along vertices

Value

A numeric vector of vertex.ids reachable from the initial set of vertex.id by 'traveling' forward in time along active vertices and edges subject to bounding paramters.

Note

This is a pure R implementation, probably very slow.

This function could be ill-defined when using non-Inf `per.step.depth` with networks with instantaneous (onset=terminus) spells as it will treat elements active at time t as active until the next change in the network.

TODO: should be able to specifiy edge weight attribute to be included in time calcualtions.

Author(s)

skyebend

See Also

See also `tPath` for a dramatically faster implementation

tDegree *Report momentary degree of a networkDynamic object at multiple timepoints*

Description

Calculates the degree of vertices at a sequence of time points over a network's temporal evolution

Usage

```
tDegree(nd, start, end, time.interval = 1, cmode = c("freeman", "indegree", "outdegree"))
```

Arguments

nd	the networkDynamic object to be evaluated
start	optional numeric time value at which evaluation should start (default is first observed time)
end	optional numeric time value at which evaluation should end (default is last observed time)
time.interval	optional numeric value giving time interval between evaluations (default is 1)
cmode	mode for evaluating degree. one of "freeman", "indegree", "outdegree"

Details

Evaluates the momentary degrees of a network at multiple time points and returns results in a form suitable for sumarizing the distributions. If a vertex is not active at a time point, its degree will be recorded as NA.

Value

A ts (time series) object, a numeric matrix with giving the momentary degree of each vertex at each time point. Columns coresponding to each vertex in the input network and row corresponding to each time point at which degree was evaluated.

Author(s)

skyebend

See Also

See also tSnaStats(nd,'degree') and tErgmStats(nd,'sociality') for alternate ways to compute degree using external packages.

Examples

```
data(McFarland_cls33_10_16_96)
tDegree(cls33_10_16_96)
# compute mean temporal degree
mean(tDegree(cls33_10_16_96),na.rm=TRUE)
## Not run:
library(networkDynamicData)
data(concurrencyComparisonNets)
# compute mean for each network, sampled at 11 time points
mean(colMeans(tDegree(base,start = 0,end=102,time.interval = 10)))
mean(colMeans(tDegree(middle,start = 0,end=102,time.interval = 10)))
mean(colMeans(tDegree(monog,start = 0,end=102,time.interval = 10)))

# plot distribution of vertices' mean momentary degree
hist(rowMeans(tDegree(base,start = 0,end=102,time.interval = 10)))

# plot distribution of momentary degrees of vertices
hist(tDegree(base,start = 0,end=102,time.interval = 10))

## End(Not run)
```

`temporal density` *Functions to compute temporal density-related measures on dynamic networks*

Description

These functions provide various network-level statistics giving information on the fraction of time edges are active in `networkDynamic` objects

Usage

```
tEdgeDensity(nd,mode=c('duration','event'),
             agg.unit=c('edge','dyad'),active.default=TRUE)
```

Arguments

`nd`	a `networkDynamic` object to evaluate density on
`mode`	option indicating if `'duration'` of edge spells should be considered or only the `'event'` count (for networks in which events have zero-durations)
`agg.unit`	option indicating how to calculate the possible observable time to be used as the denominator: `'edge'` only counts existing edges `'dyad'` counts all possible dyads.
`active.default`	logical, default TRUE. should edges without explicit timing information be considered active by default?

Details

The `tEdgeDensity` measure by default (`mode='duration', agg.unit='edge'`) computes the total duration of activity of all the edges in the network and divides by the total amount of observable time for all the dyads *between which edges are ever observed*. Can be interpreted as the average fraction of observed edges active at any time. A value of 1 corresponds to a network in which all of the observed edges are always active (but the network still may be topologically sparse, having a low density)

The `tEdgeDensity` function with `mode='event'` computes the number of events (spells) occurring on each edge the network and divides it by the total amount of observable time per dyad ever observed to have an edge within the time bounds of the network. Can be interpreted as the fraction of existing ties toggling in a unit time step?

The `agg.unit='dyad'` measure computes the total duration (or count of events) of activity of all the edges in the network, and divides by the total amount of observable time for all the *possible dyads* (existing and non-existing edges). Value of 1 corresponds to a fully-connected network in which all edges are always active, value of 0 would be a network with no active edges. Can be interpreted as the average fraction of possible edges active at any time.

For networks with instantaneous spells, the event measures would be used in preference to the duration measures, as all of the events will have zero durations.

Note that all of these measures depend on having an accurate value for the temporal bounds of the network. If a `net.obs.period` exists, it will determine the range of observations. If it does not exist, the range will be the (non-Inf) range of earliest and latest events found on the network by `get.change.times`. If no non-Inf range exists (presumably because all ties are always active or always inactive), the range (0-1) will be used.

For sensible results on discrete networks, the measures are effectively making the assumption that the time increment is 1. TODO: read time increment from net.obs.period if it exits?

Networks with no edges or vertices will return 0, although they are technically undefined.

Behavior with multiplex ties? dyad measures could range above 1, edge measures will re-normalize.

Value

A numeric value representing the network-level measure of the density metric applied

Note

These are experimental functions, names and arguments still subject to change. Should these be collapsed to a single measure with multiple arguments?

Author(s)

skyebend

References

none yet

Examples

```
## Not run:
require(networkDynamicData)
data(hospital_contact)
tEdgeDensity(hospital)

## End(Not run)
```

`tErgmStats` *Calculate network summary statistics at multiple time points*

Description

Applies a ergm-style formula of network statistics to cross-sectional networks collapsed from a networkDynamic at multiple time points to construct a matrix of values describing the change in statistics over time.

Usage

```
tErgmStats(nd, formula, start, end, time.interval = 1, aggregate.dur, rule)
```

Arguments

`nd`	`networkDynamic` object to be evaluated
`formula`	a character string providing an ergm term name or the 'right hand side' of an ergm formula. For example `'~ edges + concurrent'`
`start`	optional numeric time value at which evaluation should start (default is first observed time)
`end`	optional numeric time value at which evaluation should end (default is last observed time)
`time.interval`	optional numeric value giving time interval between evaluations (default is 1)
`aggregate.dur`	optional numeric value giving the duration of time bin to aggregate over for each evaluation (default 0). See `network.collapse`
`rule`	character vector describing rule to be used if multiple attribute values are encountred when using non-zero `aggregate.dur`. Default is `latest`. See `network.collapse` for details

Details

Constructs a set of times to evaluate based on `start`,`end` and `time.interval`. Extracts a static network at each time point and uses it to construct a formula with `f`. The formula is passed to ergm's `summary_formula` function to calculate the net value of the change statistics for each term in the formula. The values of the statistics are grouped into a time-series object (class `ts`). The ts object can be thought of as a matrix such that each column is a formula term and each row is the time point at which the statistics were evaluated. See `ergm-terms` for a list of available term statistics.

The aggregate.dur can be used to specifiy the duration of the aggregation bin, especially useful when working with continuous time networks. Usually the time.interval would be set to the same value to ensure non-overlapping bins.

Be aware that if the network's vertex activity dynamics imply cross-sectional networks of different sizes, the interpretation of the statistic at each time point may not be the same.

Value

A time-series (ts) object containing term statistics in which each column corresponds to a statistic and each row is the time point at which the statistic was evaluated

Author(s)

skyebend@uw.edu

See Also

See also summary_formula and ergm-terms. For more information about time-series objects, see ts and plot.ts for plotting quickly plotting timelines for multiple statistics. The summary_formula.networkDynamic function in the tergm package offers very similar functionality.

Examples

```
## Not run:
data(windsurfers)
tErgmStats(windsurfers,'~edges+degree(c(1,2))')
library(networkDynamicData)
data(concurrencyComparisonNets)
tErgmStats(base,'~edges+concurrent',
              start=0,end=100,time.interval = 10)
# show as multiple plots
plot(
  tErgmStats(base,'~edges+concurrent',
               start=0,end=100,time.interval = 10),

               )

## End(Not run)
```

---

tiedDuration *Compute the duration of time (or count of events) that each vertex is tied/connected to others by an edge*

Description

Computes the total duration that each vertex in the network is tied to other vertices by incident edges. Alternately, if `mode="counts"`, computes the total number of incident edge spells each vertex is tied by. The later is especially useful for continuous time networks tied by edges with 0-duration events. For directed networks, the durations can be filtered using the `neighborhood` argument to include only incoming, outgoing, or all ties combined in order to return out-tiedor in-tied durations.

Usage

```
tiedDuration(nd, mode = c("duration", "counts"),
                     active.default = TRUE,
                     neighborhood = c("out", "in", "combined"))
```

Arguments

`nd`	a `networkDynamic` object describing the network for which durations should be calculated
`mode`	either `"duration"` or `"count"` indicating if the sum of edge durations or the count of the number of edge events should be returned
`active.default`	logical, should edges with no defined activity spells be considered active by default?
`neighborhood`	value of `"out"`, `"in"` or `"combined"` indicating if – for directed networks – the durations (counts) should be limited to the vertices' ties outgoing, incoming, or combined (both).

Details

Implemented internally using the `as.data.frame.networkDynamic` function and so follows the same truncation conventions for handling censored edges (edges that are active before or after the observation window of the network)

Value

a numeric vector of length equal to the number of vertices in the network with a value equal to the sum of durations (or counts) of active edges incident upon the vertex.

Note

Should the default neighborhood for directed network be 'combined'?

Author(s)

skyebend@uw.edu

Examples

```
data(moodyContactSim)
tiedDuration(moodyContactSim)

data(McFarland_cls33_10_16_96)
# compute ratio of incoming vs. outgoing speach acts
outDur <- tiedDuration(cls33_10_16_96, mode='counts',neighborhood = 'out')
inDur <- tiedDuration(cls33_10_16_96, mode='counts',neighborhood = 'in')
outDur / inDur
```

`timeProjectedNetwork` *Construct a time-projected ("multi-slice") network by binning a networkDynamic object*

Description

Builds a new static representation of a dynamic network constructed by binning the dynamic network into static slices and constructing new directed 'identity-arcs' between the vertices' realizations in successive time slices.

Usage

```
timeProjectedNetwork(nd, start = NULL, end = NULL,
                        time.increment = NULL, onsets = NULL, termini = NULL,
                        ...)
```

Arguments

`nd`	the `networkDynamic` object for which the time projected network should be constructed
`start`	optional numeric start time to be use as lower bound for binning interval (default is what is observed in network)
`end`	optional numeric end time to be use as upper bound for binning interval (default is what is observed in network)
`time.increment`	value for the offset (and duration) between successive samples. Will default to 1 if not otherwise specified
`onsets`	A numeric vector containing the onset times of the networks to be extracted. This must be accompanied by `termini` of the same length.
`termini`	A numeric vector containing the terminus times of the networks to be extracted. This must be accompanied by `onsets` of the same length.
`...`	Additional arguments to `network.collapse` (such as `rule`,`active.default`

Details

Uses network.collapse to bin the nd network into a list of static networks, aggregates them into a new network with size equal to original network size X number of slices. To assist with plotting, an edge attribute edge.type is added to all of the edges, having the value 'within_slice' for edges existing the the original network and 'identity_arc' for edges linking the vertices in time. Vertex attributes (possibly collapsed TEAs) are copied from the original network to the projected network. Because of the assumed directionality of time, the output network will always be directed, with the identity arcs pointing forward in time. If the input network is undirected, two corresponding directed edges (one in each direction) will be added in the projected network. Edge attributes (possibly collapsed TEAs) will be copied from the original network to the corresponding within-slice edges in the projected network.

Vertex activity is currently ignored in the projected network (retain.all.vertices is set to TRUE internally to force all time slice networks to have the same size).

As with all discrete representations of dynamic processes, the time projected graph is an approximation and may over- or under-represent some transmission potential depending on the choice of bin size. Binning is performed by get.networks, so will use its defaults if not specified.

Value

a network object that encodes a discrete time representation of the temporal evolution of the input networkDynamic object.

Author(s)

Skye Bender-deMoll (skyebend@uw.edu), James Moody

References

James Moody (2015) Static Representations of Dynamic Networks (DRAFT)

Earlier citations?

See Also

network.collapse

Examples

```
data(moodyContactSim)


# use slices at each changing time point
library(networkDynamicData)
data(vanDeBunt_students)
times<-get.change.times(vanDeBunt_students)
vanDProj<-timeProjectedNetwork(vanDeBunt_students,onsets = times,termini = times)
# plot it with gray for the time edges
plot(vanDProj,
    arrowhead.cex = 0,
     edge.col=ifelse(vanDProj%e%'edge.type'=='within_slice','black','gray'),
```

```
     vertex.cex=0.7,mode='kamadakawai')
## Not run:

# compute shortest temporal path distances from each vertex in first slice
# to each vertex in last slice
library(sna)
geodist(vanDProj)$gdist[1:32,193:224]

# bin the moody sim into 100 timestep chunks
# (this will over-represent some transmission potential)
moodyProj<-timeProjectedNetwork(moodyContactSim,time.increment=100)
plot(moodyProj,arrowhead.cex = 0,
    edge.col=ifelse(moodyProj%e%'edge.type'=='within_slice','black','gray'),
    vertex.cex=0.7,displaylabels = TRUE,label.cex=0.6)

## End(Not run)
```

---

tReach *computes sizes of temporally reachable sets in a dynamicNetwork*

Description

computes sizes of temporally reachable sets in a dynamicNetwork, using either a full census or sample of starting vertices of the specified size

Usage

```
tReach(nd, direction = c("fwd","bkwd"), sample=network.size(nd),
      start, end,  graph.step.time=0)
```

Arguments

nd	a networkDynamic object
direction	currently only 'fwd' works, to calculate the sizes forward-reachable sets (as opposed to backwards-reachable)
sample	numeric, indicates the size of the sample of vertices to use to calculate the reachable sets
start	optional numeric start time to begin path search
end	optional numeric time to end path search
graph.step.time	How much time should be added for each edge traversal (graph hop)? See tPath for details.

Details

tReach calls link{tPath} on with each starting vertex to determine the sizes of the sets of vertices that are reachable. If sample is set to something less than the size of the network, it will sample that many vertices instead of doing the (expensive) full census. Note that when the vertices are chosen as a sample, results may vary between calls to this function

Value

a vector of length equal to sample giving the sizes of the set of vertices reachable from each seed vertex within the specified time bounds

Note

Needs implementation of backwards sets

Author(s)

skyebend

See Also

See also as tPath

Examples

```
data(moodyContactSim)
tReach(moodyContactSim)

# only sample 3 paths
tReach(moodyContactSim,sample=3)

# what fraction of the network could each vertex reach?
tReach(moodyContactSim)/network.size(moodyContactSim)

# what fraction of the network could each vertex be reached by?
tReach(moodyContactSim,direction='bkwd')/network.size(moodyContactSim)
```

---

tsna *Tools for Temporal Social Network Analysis*

Description

Temporal SNA tools for continuous- and discrete-time longitudinal networks. having vertex, edge, and attribute dynamics stored in the networkDynamic format. This work was supported by grant R01HD68395 from the National Institute of Health.

Details

This package provides tools for working with longitudinal network data in networkDynamic-package format. This data structure is essentially a list in the network format in which elements also have an attached activity.attribute, a matrix of spells indicating when vertex or edge is active. The networkDynamic package provides tools (networkDynamic) for translating longitudinal data from various formats (timed edge lists, lists of toggles, sets of matrices, etc).

Currently the package consists of several groups of functions

- wrappers for 'static' social network analysis metrics and apply them at multiple time points
- functions that construct or use temporal paths through networks
- basic tools for measuring durations of ties, rates of change, etc
- utility functions for plotting, etc

The sections below provide some additional details. The package vignette (`browseVignettes(package='tsna')`) gives additional examples and illustrations of key concepts.

It is the intention that, like the networkDynamic package, tsna should support both continuous time and discrete time representations of networks. However, we are prioritizing development of discrete time measures suitable for use with simulation data (i.e. stergm models) so many of the functions are still missing the necessary arguments to facilitate binning.

Standard 'static' Sna metrics

These functions operate by collapsing the dynamic network into a static network at a series of regular intervals and returning the results as a time series ts object. They can provide general description of trends in a network dataset. Generally assumes that vertex set is not substantially changing.

- tErgmStats – descriptive stats (ergm terms) from the ergm package
- tSnaStats – descriptive stats from the sna package. Both graph- and vertex-level measures. centralities, components, reciprocity, betweenness, triad-census, etc.

Temporal path based metrics

These functions compute and use temporal paths (network geodesics that are constrained by the activity times of edges) through a network.

- tPath
- tReach

Rates and Duration

These functions can be used to compute distributions of (observed) activity durations in a data structure. Note that due to censoring (edges that begin before or end after the time observation window for the network) the observed durations may be biased away from the 'real' values (or model parameters). The duration estimate functions use various types of survival analysis to return estimates of these values.

- edgeDuration

- vertexDuration
- tiedDuration
- tEdgeDissolution
- tEdgeFormation
- tiedDuration

tsna Utilities

- as.network.tPath
- plotPaths
- timeProjectedNetwork

Data Sets

- moodyContactSim
- additional useful datasets provided by the networkDynamicData and networkDynamic packages

Source

This package is part of the statnet suite of packages http://statnet.org. For citation information see citation('tsna').

---

tSnaStats *Apply the sna package's static graph- and vertex-level network descriptive statistics at multiple time points*

Description

Samples collapsed static networks at regular intervals along a network dynamic object, applies the named static sna descriptive statistic function to each network, and returns the result as a time series. Additional arguments to the function can be included via ... . Set the sna function's directedness and self-loops flags appropriately by default.

Usage

```
tSnaStats(nd, snafun, start, end, time.interval = 1, aggregate.dur=0, rule='latest', ...)
```

Arguments

nd	a networkDynamic object to be evaluated
snafun	character string giving the name of the sna package function to be applied. i.e 'mutuality'
start	optional numeric time value at which evaluation should start (default is first observed time)

end	optional numeric time value at which evaluation should end (default is last observed time)
time.interval	optional numeric value giving time interval between evaluations (default is 1)
aggregate.dur	optional numeric value giving the duration of time bin to aggregate over for each evaluation (default 0). See network.collapse
rule	character vector describing rule to be used if multiple attribute values are encountred when using non-zero aggregate.dur. Default is latest. See network.collapse for details
...	additional arguments to be passed on to the sna function. See docs for each function for possible arguments.

Details

This wrapper directly calls functions in the sna package, so it will only work if that package is installed. Below is a list of supported functions:

Graph-Level statistics:

- components Number of (Maximal) Components Within a Given Graph
- triad.census Davis and Leinhardt Triad Census
- connectedness Graph Connectedness Scores
- dyad.census Holland and Leinhardt MAN Dyad Census
- efficiency Graph Efficiency Scores
- gden Graph Density
- grecip Graph Reciprocity
- gtrans Graph Transitivity
- hierarchy Graph Hierarchy Scores
- lubness Graph LUBness Scores
- mutuality Graph Mutuality
- centralization Graph Centralization (must provide centrality measure)

Vertex-level statistics:

- closeness Vertex Closeness Centrality Scores
- betweenness Vertex Betweenness Centrality Scores
- bonpow Vertex Bonacich Power Centrality Scores
- degree Vertex Degree Centrality Scores
- evcent Vertex Eigenvector Centrality Scores
- flowbet Vertex Flow Betweenness Scores
- graphcent Vertex (Harary) Graph Centrality Scores
- infocent Vertex Information Centrality Scores
- loadcent Vertex Load Centrality Scores
- prestige Vertex Prestige Scores

Most of the sna functions involve converting the network to a matrix and can be quite expensive to calculate for a single time point, so use care when applying to large or long-duration networks.

Some of the sna functions are undefined or produce numerical errors when applied to networks with certain configurations (such as zero edges).

The sna functions generally cannot handle networks with no vertices, so stats will be replaced with NA when they are encountered.

There may be some overlap with ergm terms available through tErgmStats and the ergm version will generally be faster

Value

a ts (time series) object. A matrix in which rows correspond to the time points evaluated and columns correspond to values of statistics produced. In the case of vertex-level indices, there will be one column per vertex. For the census measures, each column will correspond to a census element.

Note

Note that this is an early DRAFT implementation. Does not yet include binning options needed for non-discrete time networks, and has not been tested with networks that have changing vertex activity.

Author(s)

skyebend@uw.edu

References

Carter T. Butts (2014). sna: Tools for Social Network Analysis. R package version 2.3-2. http://CRAN.R-project.org/package=sna

See Also

See also tErgmStats,

Examples

```
library(networkDynamicData)
data(harry_potter_support)

# compute triad census scores for each time point
tSnaStats(harry_potter_support,snafun='triad.census')

# compute graph transitivities
tSnaStats(harry_potter_support,snafun='gtrans')
## Not run:
data(concurrencyComparisonNets)
# since thes are big nets, with lots of timepoints,
# set time.interval to avoid evaluating every step
tSnaStats(base,'prestige',time.interval=25,rescale=TRUE)
```

```
# since it is time series, easy to plot
plot(tSnaStats(base,'components',time.interval=10))

## End(Not run)
```

Index