Package 'ndtv'

November 20, 2022

Type Package

Title Network Dynamic Temporal Visualizations

Version 0.13.3

Date 2022-11-20

Depends R (>= 3.0), network (>= 1.13),networkDynamic (>= 0.9),animation (>= 2.4),sna

Imports MASS, statnet.common, jsonlite, base64

Suggests tergm (>= 3.6), ergm, tsna, testthat, knitr, htmlwidgets, scatterplot3d

Description Renders dynamic network data from 'networkDynamic' objects as movies, interactive animations, or other representations of changing relational structures and attributes.

License GPL-3 + file LICENSE

URL https://github.com/statnet/ndtv

NeedsCompilation no

Author Skye Bender-deMoll [cre, aut], Martina Morris [ctb]

Maintainer Skye Bender-deMoll <skyebend@uw.edu>

Repository CRAN

Date/Publication 2022-11-20 22:10:05 UTC

R topics documented:

`ndtv-package` *Network Dynamic Temporal Visualization (ndtv)*

Description

Construct visualizations such as timelines and animated movies of networkDynamic objects to show changes in structure and attributes over time.

Details

For version and license information, run `packageDescription('ndtv')`

Uses network objects with dynamics encoded using networkDynamic.

Key features:

- Compute a dynamic layout using compute.animation.
- Render it as a movie using render.animation.
- Render animations to a web page using render.d3movie
- Plot multiple 'stills' of a movie with filmstrip
- Plot a timeline of edge and vertex activity with timeline
- Plot network geodesic proximities as a stream graph with proximity.timeline

To view package vignettes and extended examples, see `browseVignettes(package='ndtv')`.

For a more indepth tutorial, see `https://statnet.org/Workshops/ndtv_workshop.html`

The package includes several example datasets:

- `msm.sim` output of a stergm simulation of basic sex contact network model
- `short.stergm.sim` Very Very Basic stergm simulation output (25 time steps)
- `stergm.sim.1` Very Very Basic stergm simulation output (100 time steps)
- `toy_epi_sim` Toy Epidemic Simulation Output from the EpiModel package

Report bugs at: `https://github.com/statnet/ndtv/issues`

Author(s)

Skye Bender-deMoll, and the Statnet Commons team Maintainer: Skye <skyebend@uw.edu>

References

Bender-deMoll, S., Morris, M. and Moody, J. (2008) Prototype Packages for Managing and Animating Longitudinal Network Data: dynamicnetwork and rSoNIA *Journal of Statistical Software* 24:7.

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). doi:10.18637/jss.v024.i03.

Butts CT (2008). network: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). doi:10.18637/jss.v024.i02.

Carter T. Butts, Ayn Leslie-Cook, Pavel N. Krivitsky and Skye Bender-deMoll (2015). networkDynamic: Dynamic Extensions for Network Objects. R package version 0.7. https://statnet.org

Skye Bender-deMoll and McFarland, Daniel A. (2006) The Art and Science of Dynamic Network Visualization. *Journal of Social Structure. Volume 7, Number 2* https://www.cmu.edu/joss/content/articles/volume7/deMollMcFarland/

See Also

networkDynamic, compute.animation,render.animation for examples, and the package vignette vignette(package='ndtv').

`compute.animation` *Compute a sequence of vertex layouts over time suitable for rendering an animation.*

Description

Steps through a networkDynamic object and applies layout algorithms at specified intervals, storing the calculated coordinates in the network for later use by the render.animation function. Generally the layout are done in a sequence with each using the previously calculated positions as initial seed coordinates in order to smooth out the resulting movie. Not all network layout algorithms give good results.

Usage

```
compute.animation(net, slice.par = NULL, animation.mode = "kamadakawai",
                  seed.coords = NULL, layout.par = list(),
                  default.dist = NULL, weight.attr = NULL, weight.dist=FALSE,
                  chain.direction=c('forward','reverse'),
                  verbose = TRUE,...)
```

Arguments

net	A networkDynamic network object describing the temporal evolution of a network.
slice.par	A list of parameters which specify the time steps and aggregation that should be used when moving through the network. Example: `slice.par=list(start=0,end=100,interval=1, aggregate.dur=1,rule='latest')` The parameters are: • `start` The time point at which the sequence of layouts should begin • `end` The time point at which the sequence of layouts should finish • `interval` The amount of time between successive layouts • `aggregate.dur` The duration of time over which the network should be aggregated to derive the network for each layout • `rule` The aggregation rule to be used when collapsing the network.
animation.mode	The name of the network animation layout to be used. These layouts are name network.layout.animate.something but will be matched using the final part of the name. Current useful values are: • `network.layout.animate.kamadakawai` essentially wrapper for the Kamadakawai layout included in the network package. • `network.layout.animate.MDSJ` a wrapper to do a Stress Majorization optimized MDS layout using the Multi Dimensional Scaling for Java package. Note, due to license restrictions, this algorithm is for non-commercial use only) • `network.layout.animate.useAttribute` applies coordinates stored in a user-generated dynamic network attribute • `network.layout.animate.Graphviz` a wrapper for the Graphviz software library –if the library is installed on your system.
seed.coords	(optional) an array of initial positions to be used for the very first layout in the sequence
layout.par	A list of parameters to be passed to the layout algorithm.
default.dist	The default distance to be used to separate nodes (or disconnected network components). Default to `sqrt(network.size(net))`. See `layout.distance`.
weight.attr	charater providing the name of a (possibly dynamic) numeric edge attribute defining weights for the edges in each time slice. The values `activity.duration` or `activity.count` can be used to weight edges by the duration or count of the edge's activity spells in the time slice.
weight.dist	logical, defaults to `FALSE`, meaning that the edge weight values provided by `weight.attr` will be treated as similarities (larger values means closer). A value of `TRUE` means that weights should be intrepreted as distances. See `layout.distance` for more information.
chain.direction	a value of `'forward'` indicates the chain of layouts should be computes in forward temporal order. A value `'reverse'` runs the chain backwards. For some layouts, reverse-chaining means that isolated vertices are more likely to have positions close to the partners they will be tied to.

verbose	If true, additional information about the layout process and progress will be returned to console.
...	possible additional arguments to be passed to sub processes

Details

This function is under active development so implementation and parameters will continue to change.

Value

Invisibly returns original network argument (which is also modified in-place), with the addition of a network variable `slice.par` storing the slice parameters used, and dynamic node attributes `animation.x` and `animation.y` storing the coordinates calculated for each time point.

Author(s)

Skye Bender-deMoll, and the statnet team.

References

See docs for specific layout functions.

Bender-deMoll, S., Morris, M. and Moody, J. (2008) Prototype Packages for Managing and Animating Longitudinal Network Data: dynamicnetwork and rSoNIA *Journal of Statistical Software* 24:7.

Krivitsky P and Handcock M (2012). *Fit, Simulate and Diagnose Models for Network Evoluation based on Exponential-Family Random Graph Models*. Version 3.0-999. Project home page at https://statnet.org, https://cran.r-project.org/package=tergm.

Butts CT (2008). network: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). doi:10.18637/jss.v024.i02.

Skye Bender-deMoll and McFarland, Daniel A. (2006) The Art and Science of Dynamic Network Visualization. *Journal of Social Structure. Volume 7, Number 2* https://www.cmu.edu/joss/content/articles/volume7/deMollMcFarland/

See Also

See also `layout.distance`, `render.animation`, `network.layout.animate.MDSJ`,ndtv, package vignette (`vignette('ndtv')`) for examples.

`effectFun` *functions to manipulate graphic attributes of network for 'special effects'*

Description

Functions that can return appropriate graphic attributes (i.e. color interpolation) based on properties of the network (ages of edges,etc)

Usage

```
effectFun(name, ...)
effect.edgeAgeColor (net, onset, fade.dur,
                     start.color = "#000000FF", end.color = "#00000000",
                     na.color = "#CCCCCC55")
effect.vertexAgeColor (net, onset, fade.dur, start.color = "#000000FF",
                       end.color = "#00000000", na.color = "#CCCCCC55")
```

Arguments

`name`	the short name of the effect function to be returned. i.e `'edgeAgeColor'` will return the `effect.edgeAgeColor` function
`...`	additional arguments to be passed in to effect functions
`net`	a network object to be evaluated
`onset`	the time at which the network should be evaluated
`fade.dur`	(effect property) numeric value giving the color duration of the interpolation
`start.color`	(effect property) color name for color value to be used at start of interpolation
`end.color`	(effect property) color name for color value to be used at start of interpolation
`na.color`	(effect property) default color name for color value to be used for edge/vertices that are not currently active

Details

The special effects functions can be called directly for use as graphic parameters with standard network plots, or via `effectFun` which will return the effect in a functional form so that it can evaluated/substituted at each time point as plot control function to `render.animation`

`effect.edgeAgeColor` calculates the edge of each edge in `net` at the time `onset` and uses the value to return a color interpolated between `start.color` and `end.color` by comparing the time of each edge to the `fade.dur` parameter

`effect.vertexAgeColor` does the same, but for vertices.

Users can also define functions to be called in this way

Value

effectFun returns the function named by its first argument, with any arguments matching in ... substituted.

Author(s)

skyebend@uw.edu

Examples

```
library(ndtv)
data('short.stergm.sim')
# render a plot with edges colored by age at time 24
# edges labeld with age
plot(short.stergm.sim,edge.col=effect.edgeAgeColor(short.stergm.sim,
                                                   fade.dur=25,
                                                   start.color = 'red',
                                                   end.color='blue',
                                                   onset=24),
     edge.label=edges.age.at(short.stergm.sim,24),
     edge.lwd=5)

## Not run:
# render an animation where edges are colored dynamically by their age
# starting out red and fading to blue
compute.animation(short.stergm.sim,slice.par = list(start=0,
                                                   end=25,
                                                   interval=1, a
                                                   ggregate.dur=5,
                                                   rule='latest'))
render.animation(short.stergm.sim,edge.col=effectFun('edgeAgeColor',
                                                      fade.dur=5,
                                                      start.color = 'red',
                                                      end.color='blue'))

## End(Not run)
```

---

export.dot *Export a network file as Graphviz .dot formatted text file.*

Description

A crude exporter for saving out a network in the Graphviz .dot format. http://graphviz.org/content/dot-language

Usage

```
export.dot(x, file = "", coords = NULL, all.dyads = FALSE,
            vert.attrs = NULL, edge.attrs = NULL)
```

Arguments

x	The network object to be exported
file	The file name where network should be saved
coords	Optional node coordinates to include
all.dyads	FALSE, a numeric value, or a symetric matrix of distances providing the desired lengths for all dyads.. If numeric, entries are written out for all possible dyads in the network, and the numeric value will be used to fill in the values for all the dyads in the matrix not linked by an edge (see default.dist param to layout.distance). This is necessary for some uses cases, but will generate dramatically larger files and slower performance. For the matrix and numeric cases, the values will be written as Graphviz 'len' edge attributes, and the values of edge.attrs will be ignored.
vert.attrs	optional character vector listing the names of any vertex attributes of the network that should be included as attributes of the nodes in the Graphviz dot file. (e.g. 'label', 'width')
edge.attrs	optional character vector listing the names of any edge attributes of the network that should be included as attributes of edges in the Graphviz dot file. (e.g. 'weight','penwidth')

Details

A crude exporter for saving out a network in the Graphviz .dot format. http://graphviz.org/content/dot-language

Value

Returns nothing but creates a file in .dot format: http://graphviz.org/content/dot-language

Note

This is still a partial implementation focusing on edges, edge wights, and node coordinates in order to pass the information to graphViz to use it as an external layout engine rather than a renderer.

Author(s)

Skye Bender-deMoll

References

http://graphviz.org/content/dot-language

Examples

```
library(network)
net <- network.initialize(5)
net[1,] <-1
net[2,3] <-2
export.dot(net,file="testNet.dot")
```

```
# clean up file afterwords (just for testing)
file.remove("testNet.dot")
```

`export.pajek.net` *Export a network file as a Pajek .net formatted text file.*

Description

A basic tool for exporting a network as a Pajek http://pajek.imfm.si/doku.php?id=pajek .net format text file. Does not yet encode attributes, layout information or timing info.

Usage

```
export.pajek.net(net, filename)
```

Arguments

`net`	a network object
`filename`	the file where the network object should be saved

Details

.net is basically an edgelist format with sections for vertices, arcs and edges. Vertex attributes for 'label', coordinates named 'x','y','z', 'color' as 'ic' (inner color), 'shape' as a shape value will be written in the appropriate Pajek format. An edge attribute of 'weight' will be written as the edge value, 'width' as 'w' and 'color' as 'c'. See `read.paj` for reading pajek files (time info supported)

Value

A file is written out containing the vertex and edge data.

Note

This is a very minimal implementation, mostly used for testing layout algorithms. Timing information is not yet supported.

Author(s)

Skye Bender-deMoll

References

Pajek software: http://pajek.imfm.si/doku.php?id=pajek

Pajek file format documentation: http://vlado.fmf.uni-lj.si/pub/networks/pajek/svganim/1.10.7.1/PajekToSvgAnim.pdf

Examples

```
data('toy_epi_sim')
toy_epi_sim%v%'color'<-'blue'
export.pajek.net(toy_epi_sim,filename='toy_epi_sim.net')

# clean up file afterwards (just for testing)
file.remove('toy_epi_sim.net')
```

`filmstrip` *Create a 'small multiples' plot of a networkDynamic object.*

Description

Plots several frames of a network animation of a `networkDynamic` object in a single static image as a way to provide a quick visual summary of the dynamics of the network.

Usage

```
filmstrip(nd, frames = 9, slice.par, render.par, mfrow, verbose = FALSE, ...)
```

Arguments

`nd`	networkDynamic object to be plotted
`frames`	integer number of frames to extract and render
`slice.par`	optional list of parameters to control binning of network, overrides `frames` argument. See `compute.animation`
`render.par`	optional list of parameters to control rendering of network. See `render.animation`
`mfrow`	optional two-element numeric vector giving the number of rows and columns for the layout grid. See `par`.
`verbose`	boolean,(defaults to FALSE) verbose argument to be passed to `compute.animationrender.animation`/
`...`	additional arguments to be passed to `plot.network` via `render.animation`

Details

If the `networkDynamic` object does not already have animation coordinates, calls `compute.animation` to calculate coordinates for the appropriate number of frames. The `frames` argument determines the number of evenly-spaced network slices to be rendered by `render.animation` (with the normal plot recording disabled) as images on the final plot grid. Note that if the layout has coordinates pre-computed by `compute.animation`, the slices selected by the `frames` argument may not align exactly with the previously compute slices. Passing in a `slice.par` argument will overide `frames` to determine exactly which slices will be rendered.

The layout of plot grid can be changed via the `mfrow` argument.

Value

Generates plots on the active graphics device.

Note

This is a DRAFT version of the function.

Author(s)

skyebend

See Also

See also `compute.animation`, `render.animation`.

Examples

```
data(stergm.sim.1)
filmstrip(stergm.sim.1,displaylabels=FALSE)
# print an overall title for the main plot
title('stergm.sim.1 at 9 time points')

# adjust margins of individual plots to make more room
par(mar=c(1,1,1,1))
filmstrip(stergm.sim.1)
```

`install.ffmpeg` *Instructions for installing ffmpeg on various platforms*

Description

The animation package uses ffmpeg to export movies into video formats. This internal function doesn't actually install the ffmpeg library, it just gives instructions on how to do the installation – which really just point to these docs.

Usage

```
install.ffmpeg()
```

Details

Here are some all-too-brief instructions for the various platforms. After you have installed FFmpeg on your system, you can verify that R knows where to find it by typing `Sys.which('ffmpeg')` in the R terminal. You many need to first restart R after the install.

Installing in Windows:

- Download the recent 'static' build from https://ffmpeg.org/download.html
- Downloads are compressed with 7zip, so you may need to first install a 7zip decompression program before you can unpack the installer.
- Decompress the package and store contents on your computer (probably in Program Files)
- Edit your system path variable to include the path to the directory containing ffmpeg.exe

Installing on a Mac:

- Download most recent build from https://www.evermeet.cx/ffmpeg/
- The binary files are compressed with 7zip so may need to install an unarchiving utility: https://theunarchiver.com/
- Copy ffmpeg to /usr/local/bin/ffmpeg

Installing in Linux/Unix (ffmpeg or avconv):

- FFmpeg is a standard package on many linux systems. You can check if it is installed with a command like `dpkg -s ffmpeg`. If it is not installed, you should be able to install with your system's package manager. i.e. `sudo apt-get install ffmpeg` or search 'ffmpeg' in the Software Center on Ubuntu.
- Ubuntu and Debian systems may use an alternate program named "avconv" which can be installed with `sudo apt-get install libav-tools` or by searching 'libav-tools' in Ubuntu's Software Center. Verify that R knows where to find it by typing 'Sys.which('avconv')' in the R terminal. You many need to first restart R after the install. The `animation` library should automatically use 'avconv' if it sees it instead of 'ffmpeg'. If it doesn't, you can tell it to by typing `ani.options(ffmpeg='avconv')` in your R session

Value

On winddows: Will open a web browser window to the ffmpeg website and give instructions how to open this help file.

References

https://ffmpeg.org

`install.graphviz` *Instructions for installing the Graphviz libraries on various platforms*

Description

The `network.layout.animate.Graphviz` layout provides an interface for calling the various layouts provided by the Graphviz library (http://www.graphviz.org) if it is installed on your system. Since Graphviz is *not* an R package, you must manually install it on your system to get it to work.

Usage

```
install.graphviz()
```

Details

This function doesn't actually install Graphviz, it just points to these docs which give a very brief overview of how to do it on each platform.

Installing on Windows:

- download the "current stable release" installer from http://graphviz.org/download/
- run the installer
- Edit your system path variable to include the path to the directory containing the graphviz .exe files.

Installing on a Mac: It seems that there is no longer a .pkg for the mac, but it can be installed easily via homebrew

- install the brew package manager from https://brew.sh/
- from the Terminal, run `brew install graphviz`

Installing in Linux/unix:

- Graphviz is a standard package on many linux distributions. You can check if it is installed with a command like `dpkg -s graphivz`. If it is not installed, you should be able to install it with your system's package manager. i.e. `sudo apt-get install graphviz` or search 'graphivz' in the Software Center on Ubuntu.

When Graphviz is installed correctly on any platform the R command `Sys.which('neato')` should print out the path to the installed libraries.

Value

On some platforms this function will open a web browser pointing to the download page for Graphviz.

Author(s)

skyebend

References

John Ellson et.al (2001) "Graphviz – open source graph drawing tools" Lecture Notes in Computer Science. Springer-Verlag. p483-484 http://www.graphviz.org

See Also

See `network.layout.animate.Graphviz` for more details about how ndtv usees Graphviz.

`layout.center` *Functions to center and normalize the coordinates of a network plot within a window.*

Description

The `layout.center` function takes a matrix of coordinates and an x- and y-coordinate range and centers the input coordinates within the range.

The `layout.normalize` function takes a matrix of coordinates and rescales them to the range (-1,1). If `keep.aspect.ratio=FALSE`, x- and y-coords are rescaled independently.

Usage

```
layout.center(coords, xlim, ylim)
layout.normalize(coords, keep.aspect.ratio = TRUE)
```

Arguments

`coords`	two column numeric matrix of coordinates.
`xlim`	two element numeric vector giving min and max of x axis
`ylim`	two element numeric vector giving min and max of y axis
`keep.aspect.ratio`	boolean, if FALSE, x- and y-axis will be rescaled indpendently

Details

These functions are used internally, but can also be called by the user when manipulating coordinates for layouts, especially when the coordinate ranges for a sequence of layouts do not match up well. TODO: add barycenter function, and center on vertex function

Value

The input two column numeric matrix of coordinates with positions transformed.

Author(s)

skyebend

Examples

```
data(McFarland_cls33_10_16_96)
coords<-plot(cls33_10_16_96)

# center layout coords with 100 unit area
layout.center(coords,xlim=c(0,100),ylim=c(0,100))

# rescale layout coords to unit interval
layout.normalize(coords)
```

`layout.distance` *Provides a default way to convert a network into a set of euclidian distances suitable for MDS-style layout optimization.*

Description

Computes a geodesic path distance matrix for a network after symmetrizing, replacing Inf values with `default.dist`

Usage

```
layout.distance(net, default.dist = NULL,  weight.attr = NULL,
                weight.dist = FALSE)
```

Arguments

`net`	The network that the distance matrix should be computed for
`default.dist`	An (optional) value to be used to replace undefined values created by isolates and disconnected components.
`weight.attr`	character, (optional) the name of an edge attribute of `net` containing numeric values to use for edge distances.
`weight.dist`	logical, should the edge values given by `weight.attr` be interpreted as distances (larger values should place vertices farther apart) ? Default (`FALSE`) assumes values are similarities (larger values means stronger connection means vertices closer together).

Details

If no `default.dist` is provided the value `sqrt(network.size(net))` will be used. If input is similarity, it will be recoded/reversed to distances by subtracting each non-zero value from the max value of the matrix and adding the min value of the matrix. If the network is directed, the matrix will then be symmatrized to either the max value of i-j relation (if `weight.dist=FALSE`) or min value of i-j relation (if `weight.dist=TRUE`). Note that if the network is marked as undirected but includes bi-directional edges, the (i,j) value will be chosen instead of (j,i).

Value

A distance matrix assumed to be appropriate for the network.

Author(s)

Skye Bender-deMoll

Examples

```
test<-network.initialize(4)
add.edges(test,tail=1:2,head=2:3)
# in adjacency matrix form
as.matrix(test)
# as matrix of geodesic distances
layout.distance(test,1.5)
```

`msm.sim` *MSM.sim : output of a stergm simulation of basic sex contact network model*

Description

A 1000 vertex networkDynamic object that contains the output of 10 timesteps of a discrete stergm simulation of a basic sex contact network model. The model has two vertex types ('races') with different contact preferencs expressed with a `nodematch` parameter. The output network object is included as an example because re-running the model can take a while.

Usage

```
data(msm.sim)
```

Format

a `networkDynamic` object

Details

The model was built with the following stergm:

```
msm.net <- network.initialize(1000, directed=F)
msm.net %v% 'race' <- rep(c(1,2),500)
sm.form.constraints <- ~bd(maxout=2)
msm.form.formula <- ~edges+nodematch('race')+
degree(2)
msm.target.stats <- c(450,375,50)
msm.diss.formula <- ~offset(edges)+offset(nodematch("race"))
msm.theta.diss <- c(2.944, -0.747)
msm.fit <- stergm(msm.net,
formation= msm.form.formula,
dissolution= msm.diss.formula,
targets="formation",
target.stats= msm.target.stats,
offset.coef.diss = msm.theta.diss,
form.constraints=msm.form.constraints,
estimate = "EGMME",
```

```
control=control.stergm(SA.plot.progress=TRUE)
)
msm.sim <- simulate(msm.fit,nsim=1,time.slices=100)
```

However, the tergm-related output that would normally be attached to the network (toggles, etc) has been removed.

Source

statnet project stergm tutorial.

Examples

```
require(network)
require(networkDynamic)
data(msm.sim)
# show the network, aggregating 5 timesteps
plot(network.extract(msm.sim,onset=0,terminus=5),
  vertex.col=msm.sim%v%"race",vertex.cex=0.5,edge.col="gray")

# this could take 10 minutes, so don't run in example checking
## Not run:
# use ndtv to render a movie of the momentary view of the network
render.animation(msm.sim,vertex.col=msm.sim%v%'race',vertex.cex=.5)
ani.replay()
saveVideo(ani.replay(),video.name="msm.simMomentary.mp4", other.opts="-b 5000k",clean=TRUE)

# another version, this time with more temporal aggregation
msm.sim <- compute.animation(msm.sim,slice.par=list(start=0,
                                                    end=10,
                                                    interval=1,
                                                    aggregate.dur=3,
                                                    rule='latest'))
# also more render more inbetween frames
render.animation(msm.sim,render.par = list(tween.frames = 15,show.times=TRUE),
  vertex.col=msm.net%v%'race',vertex.cex=.5)
saveVideo(ani.replay(),video.name="msm.sim3Aggregated.mp4", other.opts="-b 5000k",clean=TRUE)

## End(Not run)
```

`ndtvAnimationWidget` *htmlwidgets wrapper functions for including ndtv-d3 animations in shinyapps*

Description

Wrapper functions to provide ndtv-d3 animation as an `htmlwidget`for use within an RStudio "shiny" web application. These functions are not normally called by R users directly. For example shiny app template code please see the 'server.R' and 'ui.R' files at https://github.com/statnet/ndtv/tree/master/htmlWidgetShinyTest

Usage

```
ndtvAnimationWidget(out, options, width = NULL, height = NULL)

renderNdtvAnimationWidget(expr, env = parent.frame(), quoted = FALSE)

ndtvAnimationWidgetOutput(outputId, width = "100%", height = "500px")
```

Arguments

out	the data structure describing the network animation. produced internally by render.d3movie
options	usually the 'd3.options' from render.d3movie
width	Display width for the widget. Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended.
height	Display eight for the widget. Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended.
outputId	See htmlwidgets-shiny
expr	An expression that does any necessary network processing and generates an HTML widget (usually via render.d3move). See htmlwidgets-shiny
env	The environment in which to evaluate expr. See htmlwidgets-shiny
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable. See htmlwidgets-shiny

Details

The ndtv-d3 interactive HTML5 network animation can normally be produced via render.d3movie(...,output.mode='ht
These functions are wrappers to make it possible to embed the animations as part of a 'Shiny' (https://shiny.rstudio.com/) web application.

renderNdtvAnimationWidget should be used as the wraper for the render.d3movie call within the app's server.R file and ndtvAnimationWidgetOutput is the corresponding ui component to include in the ui.R file. See htmlwidgets-shiny

ndtvAnimationWidget initializes the widget, usually called automatically inside render.d3movie when output.mode='htmlWidget'.

Author(s)

skyebend@uw.edu

See Also

htmlwidgets-package, render.d3movie

network.layout.animate *Sequentially-stable network layout algorithms suitable for generating network animations.*

Description

The network.layout.animate.* layouts are often adaptations or wrappers for existing static layout algorithms with some appropriate presets. They all accept the coordinates of the 'previous' layout as an argument so that they can try to construct a suitably smooth sequence of node positions. Usually these layouts are not called directly and instead selected by specifying the appropriate animation.mode argument to compute.animation

Usage

```
network.layout.animate.kamadakawai(net, dist.mat = NULL, default.dist = NULL,
                        seed.coords = NULL, layout.par = list(),verbose=FALSE)

network.layout.animate.MDSJ(net, dist.mat = NULL, default.dist = NULL,
                seed.coords = NULL, layout.par=list(max_iter=50, dimensions =
                 2),verbose=TRUE)

network.layout.animate.useAttribute(net, dist.mat = NULL, default.dist = NULL,
     seed.coords = NULL,  layout.par = list(x = "x", y = "y"), verbose = TRUE)

network.layout.animate.Graphviz(net, dist.mat = NULL, default.dist = NULL,
     seed.coords = NULL,  layout.par = list(), verbose = TRUE)
```

Arguments

net	The network (or temporal sub-network) that will be used to determine vertex positions.
dist.mat	A (usually optional) matrix of distances between vertices that should be used to define node positions. This is important to provide if network edge weights need special handling - for example to be fliped from similarities to distances, symmetrized, etc.
default.dist	A default distance value which a layout may use to fill in for undefined dyads to space out isolates and disconnected components.
seed.coords	A two-column by n-vertex matrix of starting coordinates for the layout to use, usually the coordinates of the previous layout.
layout.par	A list of named layout parameters specific to the algorithm.
verbose	Print more information about the layout process

Details

These layouts are generally called by compute.animation on a sequence of extracted networks, with each layout fed the output of the previous layout

Usually if the dist.mat is not included, one will be calculated using the layout.distance function which will compute the geodesic path length distance between nodes after symmetrizing the network and replacing Inf values with either sqrt(network.size) or the passed in default.dist

KamadaKawai

The KamadaKawai option provides resonably good dynamically stable layouts. It computes a symmetric geodesic distance matrix from the input network (replacing infinite values with default.dist, and seeds the initial coordinates for each slice with the results of the previous slice in an attempt to find solutions that are as close as possible to the previous positions. However, it performs poorly on large networks and is not as stable as MDSJ. See network.layout.kamadakawai for more details about the implementation and parameters

MDSJ

The MDSJ layout uses the MDSJ Java library written by Christian Pich, Algorithmics Group, Department of Computer & Information Science, University of Konstanz, Germany http://algo.uni-konstanz.de/softwar (original url), 2009. The library does Multidimensional Scaling (MDS) of the distance matrix using SMACOF optimization. Because MDSJ is released under a creative commons by-nc-sa license it is not distributed with the ndtv package, but an installer is included.

When the MDSJ layout is called it checks for working Java installation, and then checks if MDSJ is installed. If not, it prompts the user and (optionally) downloads and installs MDSJ. If MDSJ is not installed, it falls back to calling the KamadaKawai layout instead.

MDSJ is quite fast for larger networks, but relatively less efficient for smaller ones because of the overhead of system calls and Java start up for each layout. The verbose option prints more information on the Java process. The max_iter parameter sets the maximum of minimization steps the algorithm can try. In cases where it seems like the layout has not completely finished, this can be set higher. The dimensions argument sets the number of dimensions the layout should be performed in and indirectly the number of columns expected and produced for coordinate matrices.

useAttribute

The useAttribute layout makes it possible to define vertex positions using a static or dynamic vertex attributes to provide the x and y coordinates for each time step. The names of the attributes to be used are passed in via the layout.par argument. For example layout.par = list(x = "myX", y = "myY") The attribute must have values defined for each time point that the network plotted.

Graphviz

The Graphviz layout is a wrapper for the Graphviz http://www.graphviz.org software library. If the library is installed on your system (see install.graphviz), it provides a number of additional high-quality layouts. When layout is called it checks for a working Graphviz installation (falling back to KamadaKawai if Graphviz cannot be found) and writes the network to a temp file using export.dot. Then the appropriate Graphviz layout engine (default is neato) is executed via a system call, and the coordinates of the vertices are parsed from the output.

Currently, the arguments to layout.par can be used to specify the Graphviz layout engine to use (i.e. gv.engine='neato' for stress-minimized, gv.engine='dot' for hierarchal, gv.engine='fdp' for force-directed, etc) and additional command-line control parameters can be passed in via gv.args. For example, to use the 'dot' layout, but change layout rank direction to Left-Right: layout.par=list(gv.engine='dot',g

See http://graphviz.gitlab.io/doc/info/command.html. Note that Graphviz's graphic rendering parameters are not used to control network plot rendering (but they may impact layout positions).

It is also possible to pass edge attributes of the network directly through to the Graphviz .dot file by passing in the names of the attributes using `gv.edge.attrs` argument to `layout.par`. For example, `layout.par=list(gv.edge.attrs='len')` will write the value of the edge attribute 'len' to a gv attribute 'len', which would control the edge lengths when using neato or fdp http://graphviz.gitlab.io/docs/attrs/len/.

The Graphviz layout normally ignores the values in `dist.mat`, but for compatibility with other layouts, it is possible to use the values in `dist.mat` to influence Graphviz's edge length by setting `layout.par` `gv.len.mode='ndtv.distance.matrix'` instead of its default `'gv.edge.len'`. This writes out all of the possible edges to the file and will overide any other edge attributes.

Value

A two-column by n-vertex matrix of coordinates.

Note

The MDSJ algorithm can only be used for non-comercial projects as it is available under the terms and conditions of the Creative Commons License "by-nc-sa" 3.0. https://creativecommons.org/licenses/by-nc-sa/3.0/

Author(s)

Skye Bender-deMoll

References

Algorithmics Group. MDSJ: Java Library for Multidimensional Scaling (Version 0.2). Available at `http://algo.uni-konstanz.de/software/mdsj/`. University of Konstanz, 2009.

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

John Ellson et.al (2001) "Graphviz – open source graph drawing tools" Lecture Notes in Computer Science. Springer-Verlag. p483-484 http://www.graphviz.org

See Also

See Also `network.layout.kamadakawai`,`layout.distance`,`compute.animation`

`proximity.timeline` *Plot a chart of a networkDynamic object in which vertices trace out paths in time, positioned vertically so that their proximity corresponds to their relative geodesic distance at the sampled time points.*

Description

This a DRAFT version of the function, parameters are likely to change. Creates a 'phase plot' chart of vertex geodesic distance proximities overtime time, with the ability to size and color the lines corresponding to each vertex with arguments similar to `plot.network`

Usage

```
proximity.timeline(nd,  start = NULL, end = NULL, time.increment = NULL,
  onsets = NULL, termini = NULL, rule='earliest', default.dist = NULL,
  vertex.col = "#55555555", label = network.vertex.names(nd),
  labels.at = NULL, label.cex = 1,
  vertex.cex = 2, splines = -0.2, render.edges=FALSE, grid=!render.edges,
  edge.col='#00000055', edge.lwd=4,
  mode=c('isoMDS','sammon','cmdscale','gvNeato','MDSJ'),
  coords=NULL,
  draw.inactive=NULL,
  spline.style=c('default','inactive.ghost','inactive.gaps',
                 'inactive.ignore','color.attribute'),
  chain.direction=c('forward','reverse'),
  verbose=TRUE, ...)
```

Arguments

`nd`	a `networkDynamic` object to be plotted.
`start`	optional numeric value giving the time to start the network sampling to be passed to `link{get.networks}`
`end`	optional numeric value giving the time to end the network sampling to be passed to `link{get.networks}`
`time.increment`	optional numeric value to increment network sampling to be passed to `link{get.networks}`
`onsets`	optional numeric vector of sampling onset time points to be passed to `link{get.networks}`
`termini`	optional numeric vector of sampling terminus time points to be passed to `link{get.networks}`
`rule`	attribute aggregation rule (default `'earliest'`) to be passed to `network.collapse`
`default.dist`	numeric default distance parameter to space apart isolates and disconnected components. Usually defaults to square root of network size (see `layout.distance`)
`vertex.col`	either a character color value, a vector of values of length equal to the size of the network, or the name of a vertex attribute containing color values to be used to color each of the vertices splines. Note that partially transparent colors work much better than opaque colors, as it is easier to see when lines overlap. When

used with spline.style='color.attribute', vertex.col can be a function with a special limited set of arguments (see Details of render.animation) which will be evaluated *at the onset* of each segment.

labels.at — numeric value or vector of values specifying the time(s) at which vertex labels should be plotted on the splines. If NULL (default), labels will not be drawn.

label — character vector of labels for vertices or name of vertex attribute to be expanded. Default is network.vertex.names. Labels only drawn if labels.at argument has a value.

label.cex — numeric character expansion factor for vertex labels

vertex.cex — either a numeric value, a vector of values of length equal to the size of the network, or the name of a vertex attribute containing numeric values to be used to scale the width of the lines (lwd) for each vertex.

splines — numeric. value controls how tightly the splines meet their control points. A value of 0 draws straight lines and sharp corners, values less than zero cause the spline to pass through the control point, values greater than zero will approximate the point. See the shape argument of xspline.

render.edges — logical (default FALSE). Should overlapping virtical lines corresponding to the edges be drawn between the the splines corresponding to the vertices at the time points of edge onsets?

edge.col — color value or edge attribute name to be used for the edge lines if render.edges=TRUE

edge.lwd — numeric line width value or edge attribute name to be used for the width of the edge lines if render.edges=TRUE

grid — logical. if TRUE, vertical lines in the background color will be drawn at the beginning of each time slice to make it easier to determine where on the splines the positions are actually set. Usually this is not used with render.edges

mode — name of MDS algorithm to be used. Currently one of isoMDS,sammon,cmdscale

coords — optional numeric matrix of pre-computed coordinates to be used instead of the algorithm in mode. The number of matrix rows must be equal to the network size and columns equal to number of time bins implied by other arguments)

draw.inactive — DEPRECATED. see spline.style

spline.style — options to control how vertices with inactive spells or changing attribute values should be drawn:

- 'inactive.ignore' ignores activity spells and draws an unbroken spline for each vertex (fastest).
- 'inactive.gaps' leaves gaps in the splines when vertices are inactive
- 'inactive.ghost' draws faint gray dotted lines under the spline so they appear in the gaps
- 'default' does 'inactive.ignore' if there are no gaps in encountered, otherwise 'inactive.ghost'
- 'color.attribute' uses the activity spells of the vertex color TEA (indicated by the vertex.col argument) to break the splines in to color segments – ignoring the the vertices activity spells

`chain.direction`
: value of `'forward'` means that the slice layouts should be computed in temporal order, with each layout initialized with the coordinates from the previous. A value of `'reverse'` causes layouts to be computed in reverse temporal order (for some layouts, this will cause less spline crossing as vertices will tend to be closer to their final state).

`verbose`
: logical, default is TRUE, in which case status messages about the computations are printed to the console, at some speed cost

`...`
: arguments to be passed to `network.collapse` (via `get.networks`) to control how the network should be aggregated during slicing

Details

The passed network dynamic object is sliced up into a series of networks. It loops over the networks, converting each to a distance matrix based on geodesic path distance with `layout.distance`. The distances are fed into an MDS algorithm (specified by `mode`) that lays them out **in one dimension**: essentially trying to position them along a vertical line. The sequence of 1D layouts are arranged along a timeline, and a spline is drawn for each vertex connecting its positions at each time point. The idea is that closely-linked clusters form bands of lines that move together through the plot.

Currently,

- `mode='sammon'` tends to produce much equally spaced lines, making it easier to follow individual vertices, but harder to see clusters
- `mode='isoMDS'` does a better job with clusters, but in some layouts converges too soon and just produces straight lines,
- `mode='cmdscale'` does a great job with clusters, but is highly unstable (coordinates will reshuffle dramatically on nearly identical networks).
- `mode='gvNeato'` tries to do a 1D Graphviz neato layout (experimental) `network.layout.animate.Graphviz`.
- `mode='MDSJ'` tries a 1D `network.layout.animate.MDSJ` layout.

For most of the layouts it is necessary to manually adjust the default dist parameter to find a value that sufficently groups together linked clusters and spaces out isolates.

Note for RStudio users: the spline rendering seems to be much slower on RStudio's graphics device than on other graphics devices such as `x11()`.

Value

Produces a plot with horizontal splines corresponding the vertices of the network and vertical proximities approximately proportional to geodesic distance. Invisibly returns a numeric matrix of coordinates corresponding to computed positions of each vertex at each time bin. This can be passed in via the `coords` argument.

Note

This is still very much a work in progress, the 1D optimization are not very stable, especially for cmdscale

Author(s)

skyebend@uw.edu

References

Some inspirational examples here: http://skyeome.net/wordpress/?p=604

See Also

See also timeline for plotting spells of vertices and edges without proximity positioning.

Examples

```
# use the classroom interaction dataset
data(McFarland_cls33_10_16_96)

# divide the first 20 minutes of time into
# overlapping 2.5 minute bins
# and make the lines for the instructors much larger
proximity.timeline(cls33_10_16_96,
   onsets=seq(0,20,0.5),
   termini=seq(2.5,22.5,0.5),
   vertex.cex=(cls33_10_16_96%v%'type'=='instructor')*4+1,
   labels.at=16)

# load the infection sim dataset
data(toy_epi_sim)
# render a timeline with vertices colored by infection status
# show only the first 5 timesteps
proximity.timeline(toy_epi_sim,vertex.col = 'ndtvcol',
                   spline.style='color.attribute',
                   mode='sammon',default.dist=20,
                   chain.direction='reverse',
                   start=1,end=5)
```

---

render.animation *Render animations of* networkDynamic *objects as movies in various formats*

Description

Takes a network object which describes a sequence of changes in properties of a network and graphically renders it out as a sequence of plots to create an animation. Normally the coordinates determining the vertex positions at specified time points should have been calculated and stored in the network object, along with the slice.par list of parameters describing when and how the network should be divided up in time. If the coordinate data is not found, compute.animation will be called with default arguments.

Appropriate 'static' networks for each time region will be generated by network.collapse. The rendering of each frame is drawn by plot.network and most arguments are supported and are

passed through to control the details of rendering. The rendered images are stored using the animation package and can be played in the plot window (ani.replay) or saved to a movie file with saveVideo.

Usage

```
render.animation(net, render.par = list(tween.frames = 10, show.time = TRUE,
            show.stats = NULL, extraPlotCmds=NULL, initial.coords=0),
            plot.par = list(bg='white'), ani.options = list(interval=0.1),
            render.cache = c('plot.list','none'), verbose=TRUE, ...)
```

Arguments

net	The networkDynamic object to be rendered, usually containing pre-computed vertex positions as dynamic attributes.
render.par	Named list of parameters to specify the behavior of the animation. • tween.frames the number of interpolated frames to generate between each calculated network layout (default 10). • show.time If TRUE, labels the plot with onset and terminus time for each slice. • show.stats NULL, or a string containing a formula to be passed to summary.stergm to display the network statistics for the current slice on the plot. e.g. "~edges+gwesp(0,fixed=TRUE)" • extraPlotCmds NULL, or additional plot commands to draw on each frame of the animation. • initial.coords default initial coords to be assigned to vertices. Can be a two-column numeric coordinate matrix, or a numeric values to be formed into a matrix.
plot.par	list of 'high-level' plotting control arguments to be passed to par. e.g. bg for background color, margins, fonts, etc.
ani.options	list of control arguments for the animation library. For example. interval controls the delay between frames in playback, ani.dev and ani.type can be used to set non-default graphics devices for rendering (i.e 'jpeg' instead of 'png'). See ani.options
render.cache	the default value of 'plot.list' causes each frame of the animation to be cached in an internal list by the ani.record function of the animation library. This is very useful for testing and replaying animations in R's plot window, but can be very slow (or cause out-of-memory errors) for large animations. If the value is set to 'none', the plot will not be recorded (but can be saved to disk via saveVideo, see examples below) and cannot be replayed via the ani.replay() function.
verbose	If TRUE, update text will be printed to the console to report the status of the rendering process.
...	Other parameters to control network rendering. See plot.network.default. Most parameters can be set to TEA attribute names, or specified as a function to be evaluated at each timestep

Details

Most of the network plotting arguments are passed directly to plot.network.default. All of the plot.network arguments pased in via ... can be specified as a TEA or special type of function to be evaluated at each time step. For example, if there was a dynamic vertex attribute named 'wealth', it could be mapped to vertex size by providing the TEA name vertex.cex='wealth'. If the wealth value needed transformation to be an appropriate vertex size, it can be specified as a function: vertex.cex=function(slice){slice%v%'wealth'*5-3}

The arguments of plot control functions must draw from a specific set of named arguments which will be substituted in and evaluated at each time point before plotting. The set of valid argument names is:

- net is the original (uncollapsed) network
- slice is the network collapsed with the appropriate onset and terminus
- s is the slice number
- onset is the onset (start time) of the slice to be rendered
- terminus is the terminus (end time) of the slice to be rendered

A few of the plot parameters have defaults that are different from the ones given by plot.network:

- jitter defaults to FALSE to prevent unwanted bouncing
- xlim and ylim default to the ranges of the entire set of network coordinates. Although they can be set to function values for interesting effects...
- xlab defaults to a function to display the time range: function(onset,terminus){paste("t=",onset,"-",terminu It also will be overidden if show.stats is set.

If no slice.par network attribute is found to define the time range to render, it will make one up using the smallest and largest non-Inf time values, unit-length non-overlapping time steps and an aggregation rule of 'latest'.

If no dynamic coordinate information has been stored, compute.animation will be called with default values to try to do the layout before giving up.

Additional plot commands passed in via the extraPlotCmds argument will be passed to eval() after each frame is rendered and can be used to add extra annotations to the plot.

One some installations, the default output from saveVideo() (really ffmpeg) produces videos in a slightly non-standard .mp4 format that won't play in Windows Media Player or QuickTime. Users have reported that adding the argument other.opts='-pix_fmt yuv420p" to saveVideo corrects the problem. Recent versions of the animation library will include this argument by default.

To avoid performance issues with the RStudio graphics device, RStudio users will see a message that ndtv is attempting to open another type of plot window. It will try to guess a platform-appropriate device type, but specific device can be pre-specified by the user by setting the R_DEFAULT_DEVICE environment variable

Value

A sequence of plots will be generated on the active plot device. If render.cache='plot.list' the recorded plots are stored as a list in .ani.env$.images.

Note

A few of the network drawing arguments have slightly different interpretations than their plot.network equivalents:

- `xlab` will be used to display time and network statistics if those `render.par` parameters are set
- `xlim` and `ylim` will be set using max and min observed coordinate values so that all network slices will appear on the plot
- `label` if not set explicitly, will default to the vertex names for the appropriate slice network.

If the background color is transparent and not explicitly set, it will be reset to white to prevent unintentional behavior when exporting movies via ffmpeg.

Author(s)

Skye Bender-deMoll, and the statnet team.

References

Skye Bender-deMoll and McFarland, Daniel A. (2006) The Art and Science of Dynamic Network Visualization. *Journal of Social Structure. Volume 7, Number 2* https://www.cmu.edu/joss/content/articles/volume7/deMollMcFarland/

See Also

`compute.animation` for generating the movie coordinates, `ani.replay`, `plot.network` and the package vignette `vignette('ndtv')`. Also `render.d3movie` for displaying movies as interactive HTML5 animations in a web browser

Examples

```
require(ndtv)
# trivial example

triangle <- network.initialize(3) # create a toy network
add.edge(triangle,1,2)
# add an edge between vertices 1 and 2
add.edge(triangle,2,3)
# add a more edges
activate.edges(triangle,at=1) # turn on all edges at time 1 only
activate.edges(triangle,onset=2, terminus=3,
e=get.edgeIDs(triangle,v=1,alter=2))
add.edges.active(triangle,onset=4, length=2,tail=3,head=1)
render.animation(triangle)
ani.replay()

# an example with changing TEA attributes
wheel <- network.initialize(10)  # create a toy network
add.edges.active(wheel,tail=1:9,head=c(2:9,1),onset=1:9, terminus=11)
add.edges.active(wheel,tail=10,head=c(1:9),onset=10, terminus=12)
```

```
# set a dynamic value for edge widths
activate.edge.attribute(wheel,'width',1,onset=0,terminus=3)
activate.edge.attribute(wheel,'width',5,onset=3,terminus=7)
activate.edge.attribute(wheel,'width',10,onset=3,terminus=Inf)
# set a value for vertex sizes
activate.vertex.attribute(wheel,'mySize',1, onset=-Inf,terminus=Inf)
activate.vertex.attribute(wheel,'mySize',3, onset=5,terminus=10,v=4:8)
# set values for vertex colors
activate.vertex.attribute(wheel,'color','gray',onset=-Inf,terminus=Inf)
activate.vertex.attribute(wheel,'color','red',onset=5,terminus=6,v=4)
activate.vertex.attribute(wheel,'color','green',onset=6,terminus=7,v=5)
activate.vertex.attribute(wheel,'color','blue',onset=7,terminus=8,v=6)
activate.vertex.attribute(wheel,'color','pink',onset=8,terminus=9,v=7)
# render it all
render.animation(wheel,edge.lwd='width',vertex.cex='mySize',vertex.col='color')


# an example with functional attributes
set.network.attribute(wheel,'slice.par',
           list(start=1,end=10,interval=1, aggregate.dur=1,rule='latest'))
# render vertex size as betweeness
render.animation(wheel,vertex.cex=function(slice){(betweenness(slice)+1)/5})


# render it directly to a movie file without caching the plots (faster)
## Not run:
saveVideo( render.animation(wheel,edge.lwd='width',vertex.cex='mySize',vertex.col='color',
           render.cache='none') )

## End(Not run)

# simulation based example
# disabled to save time when testing
## Not run:
require(tergm)
# load in example data, results of a basic stergm sim
data(stergm.sim.1)

# (optional) pre-compute coordinates for set time range
# (optional) limit time range to a few steps to speek example
slice.par=list(start=0,end=10,interval=1, aggregate.dur=1,rule='latest')
compute.animation(stergm.sim.1,slice.par=slice.par)


# define the number of inbetween frames and a formula for stats to display
render.par<-list(tween.frames=5,show.time=TRUE,
                show.stats="~edges+gwesp(0,fixed=TRUE)")

# render the movie, with labels, smaller vertices, etc
render.animation(stergm.sim.1,render.par=render.par,
                 edge.col="darkgray",displaylabels=TRUE,
                 label.cex=.6,label.col="blue")
```

```
# preview the movie in the plot window
ani.replay()

# save the movie as mp4 compressed video (if FFMPEG installed)
saveVideo(ani.replay(),video.name="stergm.sim.1.mp4",
          other.opts="-b 5000k",clean=TRUE)

## End(Not run)
```

`render.d3movie` *Render out a web-based animation of a networkDynamic object using ndtv-d3 player app*

Description

Exports a self-contained HTML file including an SVG animation of the networkDynamic object and displays it in a web browser. See `render.animation` for details of animation construction.

Usage

```
render.d3movie(net, filename=tempfile(fileext = '.html'),
                            render.par=list(tween.frames=10,
                                            show.time=TRUE,
                                            show.stats=NULL,
                                            extraPlotCmds=NULL,
                                            initial.coords=0),
                            plot.par=list(bg='white'),
                            d3.options,
                            output.mode=c('HTML','JSON','inline','htmlWidget'),
                            script.type=c('embedded','remoteSrc'),
                            launchBrowser=TRUE,
                            verbose=TRUE,...)
```

Arguments

`net`	The network (usally `networkDynamic`) object to be rendered, usually containing pre-computed vertex positions as dynamic attributes cached by `compute.animation`
`filename`	The file name of the HTML or JSON file to be generated. Must end the proper file extension ('.json' for JSON '.html' for HTML) for correct browser display.
`render.par`	Named list of parameters to specify the behavior of the animation. • `tween.frames` the number of interpolated frames to generate between each calculated network layout (default 10). • `show.time` If `TRUE`, labels the plot with onset and terminus time for each slice.

- show.stats NULL, or a string containing a formula to be passed to summary.stergm to display the network statistics for the current slice on the plot. e.g. "~edges+gwesp(0,fixed=TRUE)"
- extraPlotCmds NULL, or additional plot commands to draw on each frame of the animation.
- initial.coords default initial coords to be assigned to vertices. Can be a two-column numeric coordinate matrix, or a numeric values to be formed into a matrix.

plot.par — list of 'high-level' plotting control arguments to be passed to par. e.g. bg main for background color, margins, fonts, etc. MANY OF THESE ARE NOT YET SUPPORTED BY THE NDTV-D3 PLAYER. See list in Details below.

d3.options — list of options to configure ndtv-d3 player app. list(animationDuration=800, scrubDuration=0, enterExitAnimationFactor=0, nodeSizeFactor=0.01, playControls=TRUE, animateOnLoad=FALSE, slider=TRUE, debugFrameInfo=FALSE, debugDurationControl=FALSE,) See Details below for explanations

output.mode — character string, one of 'HTML','JSON' or 'inline'. The first exports an HTML file with embedded javascript player app including the JSON data structure describing the animation. The second just exports the JSON data structure (for loading into an existing page). The 3rd renders the HTML inside an iframe tag and supresses all other output in an attempt to make it embedable in rmarkdown documents. The 4th generates a htmlWidget suitable for displaying in an RStudio plot window or Shiny app.

script.type — if script.type='embedded', the scripts will be embedded directly in the output html page. This option is the most portable, but will require large file sizes. If script.type='remoteSrc', only the links to the scripts will be included, so the page will require an active internet connection to play the animation.

launchBrowser — if TRUE, after exporting the file R will attempt to open it in a browser

verbose — If TRUE, update text will be printed to the console to report the status of the rendering process.

... — Other parameters to control network rendering. See plot.network.default. Most parameters can be set to TEA attribute names, or specified as a function to be evaluated at each timestep. NOT ALL PLOT PARAMS ARE IMPLEMENTED YET

Details

Animations are generated using a process nearly identical to render.animation. However, instead of using R's plotting functions and the animation library, the relevant information is cached and written into a JSON-formatted file, embedded into a web page along with ndtv-d3 player, and displayed in a web browser as an interactive HTML5 SVG animation.

The ndtv-d3 player app is not a fully-featured R plot device. It only attempts to emulate the elements of plot normally used by plot.network and it understands the graphic elements and a somewhat higher level so that it will be able to handle interaction with edge and vertex objects.

However, ndtv-d3 includes several nice features to support exploring the network:

- controller buttons for playing, pausing and stepping through the animation

- time slider for jumping and 'scrubing' to parts of the movie
- pan and zoom into the network using the mouse-wheel
- click on vertices and edges to reveal their ids or abitrary text attached using the `vertex.tooltip` and `edge.tooltip` properties
- double-click on a vertex to highlight all of the connected edges and vertices

If passed a static network, by default only a single slice will be rendered and the time slider and controllers will be disabled. For consistency with `plot.network` the static mode also supports passing in a matrix of coordinates via `coord` argument which will prevent the default call to `compute.animation`.

Another advantage of `ndtv-d3` is that it does not require installing system libraries such as `ffmpeg` to render out the movie.

The coordinates for vertex postitions are read from the `animation.x` and `animation.y` TEA attributes, normally created using `compute.animation`

The list of currently supported `plot`,`plot.network` and `par` elements is

- `xlab`: label caption below the render, on the xaxis
- `main`: main headline above the render
- `displaylabels`: should vertex labels be displayed?
- `usearrows`: should arrows be drawn on edges?
- `bg`: background color (must be html compatible? need to check this)
- `vertex.cex`: vertex expansion scale factor
- `label`: labels for vertices (defaults to vertex.names)
- `label.col`: color of vertex labels
- `label.cex`: vertex label expansion scale factor
- `vertex.col`: vertex fill colors
- `vertex.sides`: number of sides for vertex polygon (shape)
- `vertex.rot`: rotation for vertex polygon
- `vertex.border`: color of vertex border stroke
- `vertex.lwd`: width of vertex border stroke
- `edge.lwd`: width of edge stroke
- `edge.col`: edge stroke color

All of the above properties can be defined as dynamic (TEA) attributes. Noteably, curved edges, edge labels, and label positioning are not yet implemented and will be ignored. The `main` and `xlab` params will not be positioned exactly as they are in `plot`

There are a few special plot parameters that are only supported by `render.d3move`:

- `vertex.tooltip` arbitrary text or html to be displayed when a vertex is clicked (default is the vertex id)
- `edge.tooltip` arbitrary text or html to be displayed when an edge is clicked (default is the edge id)

- `vertex.css.class` properties for adding arbitrary class attributes for use in CSS styling of vertices
- `edge.css.class` properties for adding arbitrary class attributes for use in CSS styling of edges
- `vertex.label.css.class` properties for adding arbitrary class attributes for use in CSS styling of vertex labels

ndtv-d3 has its own configuration properties passed in via the `d3.options` argument list, shown below with their default properties. Values which are set to `NULL` or omitted will be set to the ndtv-d3 player defaults.

- `animationDuration=800` Duration (milliseconds) of each animation step during play or step actions
- `enterExitAnimationFactor=0` Fraction (0-1) of total step animation time that edge enter/exit animations should take
- `nodeSizeFactor=0.01` Sets default node (vertex) size, as a fraction of viewport size.
- `playControls=TRUE` Show the player controls (play, pause, step, etc)
- `animateOnLoad=FALSE` If true, animation will start playing as soon as page loads.
- `slider=TRUE` Show the time slider control
- `margin=list(x=20,y=10)` SVG margins - may be overridden when setting fixed aspect ratio
- `debugFrameInfo=FALSE`Show the slice timing info in corner
- `durationControl=TRUE`Show a control to change speed of animation under the menu in the upper right corner

Value

If `output.mode='HTML'`, a file will be generated including the necessary javascript and JSON data structures If `output.mode='JSON'`, a JSON file will be generated including a section describing all of the rendered slices, and a seperate section including the entire networkDynamic object. If `output.mode='inline'`, HTML code for an iframe elment suitible for embedding in markdown documents will be printed, all other output supressed. If `output.mode='htmlWidget'`, a `htmlwidgets` object will be returned which, will produce appropriate html when 'printed' or emeded in a Shiny app

Note

This is a very preliminary draft implementation. The animations preform poorly in the Linux Firefox browser, but are ok in Firefox on other platforms and excellent in the Chrome web browser.

Author(s)

skyebend@uw.edu

References

The github repository for the ndtv-d3 javascript library is at `https://github.com/statnet/ndtv-d3/` (which is the statnet release fork of `https://github.com/michalgm/ndtv-d3/`)

See Also

See also the ndtv-d3 vignette https://statnet.org/Workshops/ndtv-d3_vignette.html, render.animation, compute.animation.

Examples

```
# render an interactive SVG animation of short.stergm.sim and open it in a browser
data(short.stergm.sim)
render.d3movie(short.stergm.sim)

# render interactive widget in rmarkdown or RStudio plot window
render.d3movie(short.stergm.sim,output.mode='htmlWidget')

# render a static network as interactive SVG with lots of html tooltip info
data(emon)
render.d3movie(emon[[5]],
    vertex.tooltip=paste(emon[[5]]%v%'vertex.names',
      emon[[5]]%v%'Command.Rank.Score',
      emon[[5]]%v%'Sponsorship',
      sep="<br>"),
    edge.tooltip=paste('Frequency:',emon[[5]]%e%'Frequency'),
    edge.lwd='Frequency')

## Not run:


# alternate code for embeding in rmarkdown
```{r,results='asis'}
render.d3movie(short.stergm.sim,output.mode='inline')
```


End(Not run)
```

---

stergm.sim.1 *Very Very Basic stergm simulation output*

Description

Simulation from a crude stergm model based on the flobusiness network. Mostly good for testing movies 'cause it is small (16 vertices) and fast. The stergm.sim.1 network is 100 simulation steps in duration. The short.stergm.sim network is an extract of the first 25 steps of stergm.sim.1 – its shorter duration makes it more suitable for quickly testing animation techniques.

Usage

```
data(stergm.sim.1)
data(short.stergm.sim)
```

Format

A networkDynamic object containing the output of the network simulations

Details

The model used to generate the sim was:

```
require(ergm)
data("florentine")
theta.diss <- log(9)
# fit the model
stergm.fit.1 <- stergm(flobusiness,
  formation= ~edges+gwesp(0,fixed=T),
  dissolution = ~offset(edges),
targets="formation",
offset.coef.diss = theta.diss,
estimate = "EGMME" )
# simulate from the model
stergm.sim.1 <- simulate(stergm.fit.1,
                    nsim=1, time.slices = 100)
```

However, the ergm-related output that would normally be attached to the network (toggles, etc) has been removed.

Source

See tergm package tutorials.

Examples

```
data(stergm.sim.1)
range(get.change.times(stergm.sim.1))
data(short.stergm.sim)
range(get.change.times(short.stergm.sim))
```

---

timeline *Plot a timeline for the edge and vertex spells of a network*

Description

Produces a 'phase plot' or timeline showing the durations of the activity spells in a networkDynamic object. Spells are traced out horizontally, with all the activity for each element (vertex or edge) in a single row.

Usage

```
timeline(x, v = seq_len(network.size(x)), e = seq_along(x$mel),
        plot.vertex.spells = TRUE, plot.edge.spells = TRUE,
        slice.par = NULL,
        displaylabels = TRUE, e.label=TRUE, e.label.col='purple',
        edge.lwd=1,
        v.label, v.label.col='blue',
        vertex.cex=1, cex, adj=0,
        edge.col = rgb(0.5, 0.2, 0.2, 0.5),
        vertex.col = rgb(0.2, 0.2, 0.5, 0.5),
        xlab, ylab, xlim, ylim, ...)
```

Arguments

x	a networkDynamic object that will have its spells plotted.
v	numeric vector of vertex ids to include
e	numeric vector of edge ids to include
plot.vertex.spells	logical, should vertex spells be plotted?
plot.edge.spells	logical, should edge spells be plotted?
slice.par	(optional) 'slice.par' list giving network binning parameters. If included, rectangles corresponding to each bin will be plotted over the spells to indicate which spell will land in bins. The bins will be drawn with slightly darker left edge more transparent right edge to evoke the effect of a right-open interval.
displaylabels	logical, should labels be drawn for each spell
e.label	character vector of edge labels or edge attribute name. Default is edge.id
e.label.col	color name or character vector of colors for edge labels (or name of edge attribute to provide them)
v.label	character vector of vertex labels or vertex attribute name. Default is network.vertex.names
v.label.col	color name or character vector of colors for vertex labels (or name of vertex attribute to provide them)
vertex.cex	numeric scaling factor, vector of numeric scaling factors or attribute name. Translated width of line (lwd) corresponding to each vertex.
edge.lwd	numeric scale factor, numeric vector, or character edge attribute name providing a numeric value for the width of the lines corresponding to each edge. Note that this does not behave exactly as edge.lwd in plot.network as it does not perform scaling based on attribute values.
cex	text size scaling for both vertex and edge labels (see plot.default)
adj	text justification parameter (see par) for both vertex and edge labels. Labels are positioned relative to onset of spell.
edge.col	color to be used to draw lines for edge spells, or vector of color names corresponding to edges, or name of edge attribute.

vertex.col	color to be used to draw lines for vertex spells, or vector of color names corresponding to vertices, or name of vertex attribute.
xlab	x-axis label for plot
ylab	y-axis label for plot
xlim	two-element numeric vector giving the x-range (time bounds) of the plot to show. Defaults to (non-infinite) max and min time of network.
ylim	two-element numeric vector giving the y-range (effectively the number of entities) of plot to show. Defaults to an appropriate mapping of the number of entities to the available plot size.
...	additional arguments to be passed to plot subroutines. See plot.default,lines,text.

Details

When the v argument is included, edges involving vertices not in v are excluded (but the reverse is not true for the e argument). If xlim range is provided and the spells corresponding to a vertex or an edge lie entirely outside its bounds they will not be shown.

Many of the arguments correspond to arguments in plot.network but are translated to the timeline plot context. For example, vertex.cex actually controls the lwd (line width) of the lines corresponding to vertex spells. The arguments are expanded using plotArgs.network so that they should give the expansion behavior and attribute look up as plot.network

Additional plotting arguments can be passed in to modify drawing. For example, lty for line style. Vertices and edges that are never active are not included on the plot.

Value

A plot is produced.

Note

not fully implemented, would be nice to be able to pass network attribute names for properties..

Author(s)

skyebend@uw.edu

See Also

See also plot.network.

Examples

```
data(stergm.sim.1)
timeline(stergm.sim.1)

# color vertices by priorates, don't show edges
timeline(stergm.sim.1,vertex.col='priorates',plot.edge.spells=FALSE)

# show only relationships among a few vertices
```

```
timeline(stergm.sim.1,v=1:8)

# zoom in on a region of time
timeline(stergm.sim.1,xlim=c(20,40))

# label vertices with numbers
# and label edges by the tail and head vertices they link
timeline(stergm.sim.1,xlim=c(0,5),v.label=1:network.size(stergm.sim.1),
    e.label=sapply(stergm.sim.1$mel,function(e){paste(e$inl,e$outl,sep='->')}) )

# show only edge spells, hi-lite edge id 20
set.edge.attribute(stergm.sim.1,'my_color','gray')
set.edge.attribute(stergm.sim.1,'my_color','red',e=20)
timeline(stergm.sim.1,edge.col='my_color',plot.vertex.spells=FALSE)

# show binning over the edges
timeline(stergm.sim.1,slice.par=list(start=0,
                                     end=100,
                                     interval=10, aggregate.dur=5,
                                     rule='latest'),
                      plot.vertex.spells=FALSE)
```

`timePrism` *Plot a networkDynamic object as sequence of snapshots in a pseudo-3D space-time prism*

Description

Plots an image using `scatterplot3d` to render multiple network layout 'slices' in a '3D' orthographic projection in which one axis is time. The coordinates for the networks are assumed to have been generated by `compute.animation`

Usage

```
timePrism(nd, at,
              spline.v = NULL,
              spline.col = "#55555555",
              spline.lwd = 1,
              box = TRUE,
              axis = TRUE,
              planes = FALSE,
              plane.col = "#FFFFFF99",
              scale.y = 1,
              angle = 20,
              orientation = c("x", "y", "z"),
              ...)
```

Arguments

nd	a networkDynamic object to be plotted
at	a numeric vector of times at which the network should be sampled and plotted.
spline.v	optional integer vector of vertex ids to highlight with a spline linking the vertices" positions at multiple time points
spline.col	vector of colors corresponding to spline.v
spline.lwd	numeric line width for vector splines
box	a logical value indicating whether a box should be drawn around the plot to indicate its bounds
axis	a logical value indicating whether the x, y, and z axis should be drawn.
planes	a logical value indicating whether a 'plane' should be drawn to indicate the boundaries of each individual network plot
plane.col	a color value to be used to color the planes (usually partially transparent)
scale.y	numeric value giving the relative scale of y axis related to x- and z axis. (may distort network vertex shapes)
angle	numeric angle (degrees) between x and y axis (Attention: result depends on scaling).
orientation	three-element charter vector the permutation of which determines the mapping and orientation of the plot axis relative to the figure. i.e. default c('x','y','z') will place 'z' (the time dimension) 'vertically' up the page, c('z','x','y') will make the time dimension horizontal, etc.
...	additional parameters to plot.network

Details

Implements a common conceptualization of dynamic networks a series of 'layers' or 'slices' in time. Mostly useful for illustrative purposes as this plot type tends to get really crowded if more than a few network time points are shown, or vertices highlighted.

Value

invisibly returns the result of the scatterplot3d command, which contains useful functions as $xyz.convert which can be used to convert xyz coordinates into the plot space for additional annotation.

Note

Not all of the useful argument passthroughs to scatterplot3d and xspline have been implemented yet. Shapes of vertices and edges can be improperly distorted by coordinate projection.

Author(s)

skyebend@uw.edu

See Also

compute.animation,scatterplot3d, xspline, plot.network. Also filmstrip and proxmity.timeline for related static views.

Examples

```
data("short.stergm.sim")
compute.animation(short.stergm.sim)
timePrism(short.stergm.sim,at=c(1,10,20),
          displaylabels=TRUE,
          label.cex=0.5)

data(toy_epi_sim)
timePrism(toy_epi_sim,
          orientation=c('z','y','x'),
          angle=40,
          spline.v=c(7, 29, 36, 70, 82, 96),  # hilite the infected
          spline.col='red',
          spline.lwd=2,
          box=FALSE,
          planes=TRUE,
          vertex.col='ndtvcol')
```

---

toy_epi_sim *Toy Epidemic Simulation Output from the EpiModel package*

Description

An example network of a trivial simulated disease process spreading over a simulated dynamic contact network among 100 individuals for 25 discrete time steps.

Usage

```
data("toy_epi_sim")
```

Format

The format is a networkDynamic object with attached attributes for vertex.pid (persistand ids), and dynamic attributes for ndtvcol (color corresponding to infection status) and testatus (infection status of vertices)

Details

The toy_epi_sim network is example output from a basic dynamic network STERGM simulation and trivial "SI" infection simulation generated using the EpiModel package. The model had random ("edges only") edge formation and dissolution effects, with rates calculated to lead to mean edge durations of 10 time units. The infection simulation had an infection probability of 0.8.

The simulation was generated with the following code:

```
library(EpiModel)

## Network Estimation (using a tergm model)
nw <- network.initialize(n = 100, directed = FALSE)
formation <- ~ edges
target.stats <- 50
dissolution <- ~ offset(edges)
coef.diss <- dissolution_coefs(dissolution, duration = 10)
est <- netest(nw,
              formation,
              dissolution,
              target.stats,
              coef.diss,
              verbose = FALSE)

## Epidemic simulation
param <- param.net(inf.prob = 0.8)
init <- init.net(i.num = 5)
control <- control.net(type = "SI", nsteps = 25, nsims = 1, verbose =
                         FALSE)
sim <- netsim(est, param, init, control)

## Use some of EpiModel's default coloring functions to cache colors
toy_epi_sim <- get_network(sim)
toy_epi_sim <- color_tea(toy_epi_sim)
```

References

Samuel Jenness, Steven M. Goodreau and Martina Morris (2015). EpiModel: Mathematical Modeling of Infectious Disease. R package version 1.1.4. https://CRAN.R-project.org/package=EpiModel

Statnet EpiModel Tutorial http://www.epimodel.org/

See Also

See also short.stergm.sim for another basic Stergm simulation output, and msm.sim for a larger and more complex simulation without an infection process.

Examples

```
data(toy_epi_sim)
timeline(toy_epi_sim)
## Not run:


# set up layout to draw plots under timeline
layout(matrix(c(1,1,1,2,3,4),nrow=2,ncol=3,byrow=TRUE))
# plot a proximity.timeline illustrating infection spread
proximity.timeline(toy_epi_sim,vertex.col = 'ndtvcol',
                   spline.style='color.attribute',
```

```
                  mode = 'sammon',default.dist=100,
                  chain.direction='reverse')
# plot 3 static cross-sectional networks
# (beginning, middle and end) underneath for comparison
plot(network.collapse(toy_epi_sim,at=1),vertex.col='ndtvcol',
       main='toy_epi_sim network at t=1')
plot(network.collapse(toy_epi_sim,at=17),vertex.col='ndtvcol',
       main='toy_epi_sim network at=17')
plot(network.collapse(toy_epi_sim,at=25),vertex.col='ndtvcol',
       main='toy_epi_sim network at t=25')
layout(1) # reset the layout


# render an animation of the network
render.animation(toy_epi_sim,vertex.col='ndtvcol',displaylabels=FALSE)
ani.replay()

## End(Not run)
```

`transmissionTimeline` *plots network diffusion/transmission tree with generation time vs. clock/model time*

Description

Plots view of a network with positions determined by the timing and generation depth (previously calculated) in a transmission tree. The horizontal axis is model time, and the vertical axis is the number of steps from the root of the tree.

Usage

```
transmissionTimeline(x, time.attr,
                     label,
                     displaylabels = !missing(label),
                     label.cex = 0.7,
                     label.col = 1,
                     vertex.col = 2,
                     vertex.border = 1,
                     vertex.lwd = 1,
                     vertex.sides = 50,
                     vertex.cex = 1,
                     jitter=FALSE,
                     edge.col = "gray",
                     edge.lty = 1,
                     edge.lwd = 1,
                     xlab = "time",
                     ylab = "generation",
                     ...)
```

Arguments

x	an object than can be coerced into a network. The network must be a tree
time.attr	name of a vertex attribute containing the transmission/infection/diffusion time for each vertex
label	a vector of vertex labels, if desired; defaults to the vertex labels returned by network.vertex.names. If label has one element and it matches with a vertex attribute name, the value of the attribute will be used. Note that labels may be set but hidden by the displaylabels argument.
displaylabels	boolean; should vertex labels be displayed?
label.cex	character expansion factor for label text.
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.
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.
vertex.lwd	line width of vertex borders; may be given as a vector or a vertex attribute name, if vertex borders are to have different line widths.
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. NOTE: only values of 3,4,and 50 (circle) are used as they are translated to pch plot symbols.
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.
jitter	if TRUE, noise will be added to the coordinates with jitter to make overlapping vertex positions more noticeable
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.
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.
edge.lwd	line width scale for edges; May be given as a vector or edge attribute name, if edges are to have different line widths.
xlab	y-axis plot label
ylab	x-axis plot label
...	additional arguments to plot (and par)

Details

Many (but not all) of the graphical arguments to plot.network can be used and are expanded in the same way. This does not currently use the plot.network code to draw the network as non-square plot aspect ratios would cause distortion of the vertices when drawn.

Value

produces a plot, invisibly returns the coordinates of the plot.

Author(s)

skyebend@uw.edu

See Also

plot.network, proximity.timeline

Examples

```
# an edgelist describing an infection tree
el <-cbind(c(16, 13, 13,  10, 13, 16, 10, 13, 1, 10, 8, 1, 4, 4, 2, 2),
            1:16)
# a vector of infection times
infectionTimes <- c(583, 494, 634, 40, 712, 701, 224, 719,
                    674, 0, 749, 621, 453, 665, 709, 575)
# make a network object, include the infection time
infTree<-network(el,vertex.attr = list(infectionTimes),
            vertex.attrnames = list('infectionTimes'))

transmissionTimeline(infTree,time.attr='infectionTimes')
```

Index