Package ‘networkD3’

October 13, 2022

Type Package

Title D3 JavaScript Network Graphs from R

Description Creates 'D3' 'JavaScript' network, tree, dendrogram, and Sankey graphs from 'R'.

Version 0.4

Date 2017-03-18

URL https://CRAN.R-project.org/package=networkD3


License GPL (>= 3)

Depends R (>= 3.0.0)

Imports htmlwidgets (>= 0.3.2), igraph, magrittr

Suggests htmltools (>= 0.2.6), jsonlite,

Enhances knitr, shiny

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author J.J. Allaire [aut],
       Peter Ellis [ctb],
       Christopher Gandrud [aut, cre],
       Kevin Kuo [ctb],
       B.W. Lewis [ctb],
       Jonathan Owen [ctb],
       Kenton Russell [aut],
       Jennifer Rogers [ctb],
       Charles Sese [ctb],
       CJ Yetman [aut]

Maintainer Christopher Gandrud <christopher.gandrud@gmail.com>

Repository CRAN

Date/Publication 2017-03-18 17:31:09 UTC
\textbf{R topics documented:}

- networkD3-package ......................................................... 2
- as.radialNetwork ....................................................... 2
- chordNetwork ............................................................... 3
- chordNetworkOutput .................................................... 4
- dendroNetwork ............................................................. 5
- diagonalNetwork ........................................................... 7
- forceNetwork ................................................................. 9
- igraph\_to\_networkD3 ...................................................... 12
- JS .................................................................................. 13
- MisLinks ........................................................................... 14
- MisNodes .......................................................................... 15
- radialNetwork ................................................................. 15
- sankeyNetwork ................................................................. 17
- saveNetwork ..................................................................... 19
- SchoolsJournals ............................................................... 19
- simpleNetwork ................................................................. 20

\textbf{Index} \hspace{1cm} 22

\begin{verbatim}

networkD3-package  
\textit{Tools for Creating D3 Network Graphs from R}

\end{verbatim}

\textbf{Description}

Creates D3 JavaScript network, tree, dendrogram, and Sankey graphs from R.

---

\begin{verbatim}

as.radialNetwork  
\textit{Convert an R hclust or dendrogram object into a radialNetwork list.}

\end{verbatim}

\textbf{Description}

\texttt{as.radialNetwork} converts an R \texttt{hclust} or \texttt{dendrogram} object into a list suitable for use by the \texttt{radialNetwork} function.

\textbf{Usage}

\texttt{as.radialNetwork(d, root)}

\textbf{Arguments}

\begin{verbatim}
d \hspace{1cm} An object of R class \texttt{hclust} or \texttt{dendrogram}.
root \hspace{1cm} An optional name for the root node. If missing, use the first argument variable name.
\end{verbatim}
chordNetwork

Details

as.radialNetwork converts R objects of class hclust or dendrogram into a list suitable for use with the radialNetwork function.

Examples

# Create a hierarchical cluster object and display with radialNetwork
## dontrun
hc <- hclust(dist(USArrests), "ave")
radialNetwork(as.radialNetwork(hc))

---

chordNetwork

Create Reingold-Tilford Tree network diagrams.

Description

Create Reingold-Tilford Tree network diagrams.

Usage

chordNetwork(Data, height = 500, width = 500, initialOpacity = 0.8, useTicks = 0, colourScale = c("#1f77b4", "#aec7e8", "#ff7f0e", "#2ca02c", "#bbd0d2", "#04b6c5", "#7f7f7f", "#a7ff02", "#c7ff02", "#7f7f7f", "#04b6c5"), padding = 0.1, fontSize = 14, fontFamily = "sans-serif", labels = c(), labelDistance = 30)

Arguments

Data A square matrix or data frame whose (n, m) entry represents the strength of the link from group n to group m
height height for the network graph’s frame area in pixels (if NULL then height is automatically determined based on context)
width numeric width for the network graph’s frame area in pixels (if NULL then width is automatically determined based on context)
initialOpacity specify the opacity before the user mouses over the link
useTicks integer number of ticks on the radial axis. The default is ‘0’ which means no ticks will be drawn.
colourScale specify the hexadecimal colours in which to display the different categories. If there are fewer colours than categories, the last colour is repeated as necessary (if NULL then defaults to D3 colour scale)
padding specify the amount of space between adjacent categories on the outside of the graph
fontSize numeric font size in pixels for the node text labels.
chordNetworkOutput

fontFamily font family for the node text labels.
labels vector containing labels of the categories
labelDistance integer distance in pixels (px) between text labels and outer radius. The default is ‘30’.

Source


Examples

```r
## Not run:
#### Data about hair colour preferences, from

hairColourData <- matrix(c(11975, 1951, 8010, 1013,
                            5871, 10048, 16145, 990,
                            8916, 2060, 8090, 940,
                            2868, 6171, 8045, 6907),
                           nrow = 4)

chordNetwork(Data = hairColourData,
              width = 500,
              height = 500,
              colourScale = c("#000000",
                              "#FFDD89",
                              "#957244",
                              "#F26223"),
              labels = c("red", "brown", "blond", "gray"))

## End(Not run)
```

chordNetworkOutput Shiny bindings for networkD3 widgets

Description

Output and render functions for using networkD3 widgets within Shiny applications and interactive Rmd documents.

Usage

```r
chordNetworkOutput(outputId, width = "100\%", height = "500px")
renderchordNetwork(expr, env = parent.frame(), quoted = FALSE)
dendroNetworkOutput(outputId, width = "100\%", height = "800px")
```
dendroNetwork

Create hierarchical cluster network diagrams.

Usage

dendroNetwork(hc, height = 500, width = 800, fontSize = 10,
linkColour = "#ccc", nodeColour = "#fff", nodeStroke = "steelblue",
textColour = "#111", textOpacity = 0.9, textRotate = NULL,
opacity = 0.9, margins = NULL, linkType = c("elbow", "diagonal"),
treeOrientation = c("horizontal", "vertical"), zoom = FALSE)
**Arguments**

- **hc**: a hierarchical (hclust) cluster object.
- **height**: height for the network graph’s frame area in pixels
- **width**: numeric width for the network graph’s frame area in pixels
- **fontSize**: numeric font size in pixels for the node text labels.
- **linkColour**: character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
- **nodeColour**: character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
- **nodeStroke**: character string specifying the colour you want the node perimeter to be. Multiple formats supported (e.g. hexadecimal).
- **textColour**: character vector or scalar specifying the colour you want the text to be before they are clicked. Order should match the order of hclust$labels. Multiple formats supported (e.g. hexadecimal).
- **textOpacity**: numeric vector or scalar of the proportion opaque you would like the text to be before they are clicked. Order should match the order of hclust$labels.
- **textRotate**: numeric degrees to rotate text for node text. Default is 0 for horizontal and 65 degrees for vertical.
- **opacity**: numeric value of the proportion opaque you would like the graph elements to be.
- **margins**: numeric value or named list of plot margins (top, right, bottom, left). Set the margin appropriately to accommodate long text labels.
- **linkType**: character specifying the link type between points. Options are ’elbow’ and ’diagonal’.
- **treeOrientation**: character specifying the tree orientation, Options are ’vertical’ and ’horizontal’.
- **zoom**: logical enabling plot zoom and pan

**Source**

- Mike Bostock: [http://bl.ocks.org/mbostock/4063570](http://bl.ocks.org/mbostock/4063570).

**Examples**

```r
## Not run:
hc <- hclust(dist(USArrests), "ave")
dendroNetwork(hc, height = 600)
dendroNetwork(hc, treeOrientation = "vertical")
dendroNetwork(hc, height = 600, linkType = "diagonal")
dendroNetwork(hc, treeOrientation = "vertical", linkType = "diagonal")
dendroNetwork(hc, textColour = c("red", "green", "orange"))[cutree(hc, 3)],
```
diagonalNetwork

height = 600)
dendroNetwork(hc, textColour = c("red", "green", "orange")[cutree(hc, 3)],
               treeOrientation = "vertical")

## End(Not run)

diagonalNetwork

Create Reingold-Tilford Tree network diagrams.

Description

Create Reingold-Tilford Tree network diagrams.

Usage

diagonalNetwork(List, height = NULL, width = NULL, fontSize = 10,
                 fontFamily = "serif", linkColour = "#ccc", nodeColour = "#fff",
                 nodeStroke = "steelblue", textColour = "#111", opacity = 0.9,
                 margin = NULL)

Arguments

  List               a hierarchical list object with a root node and children.
  height             height for the network graph’s frame area in pixels (if NULL then height is automatically determined based on context)
  width              numeric width for the network graph’s frame area in pixels (if NULL then width is automatically determined based on context)
  fontSize           numeric font size in pixels for the node text labels.
  fontFamily         font family for the node text labels.
  linkColour         character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
  nodeColour         character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
  nodeStroke         character string specifying the colour you want the node perimeter to be. Multiple formats supported (e.g. hexadecimal).
  textColour         character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
  opacity            numeric value of the proportion opaque you would like the graph elements to be.
  margin             an integer or a named list/vector of integers for the plot margins. If using a named list/vector, the positions top, right, bottom, left are valid. If a single integer is provided, then the value will be assigned to the right margin. Set the margin appropriately to accomodate long text labels.
diagonalNetwork

Source
Mike Bostock: http://bl.ocks.org/mbostock/4339083.

Examples

## Not run:
#### Create tree from JSON formatted data
```r
## Download JSON data
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://cdn.rawgit.com/christophergandrud/networkD3/",
               "master/JSONdata//flare.json")

## Convert to list format
Flare <- jsonlite::fromJSON(URL, simplifyDataFrame = FALSE)
```

#### Create a tree dendrogram from an R hclust object
```r
hc <- hclust(dist(USArrests), "ave")
diagonalNetwork(as.radialNetwork(hc))
diagonalNetwork(as.radialNetwork(hc), fontFamily = "cursive")
```

#### Create tree from a hierarchical R list
For an alternative structure see: http://stackoverflow.com/a/30747323/1705044
```r
CanadaPC <- list(name = "Canada", children = list(list(name = "Newfoundland",
                   children = list(list(name = "St. John's"))",
                   list(name = "PEI",
                   children = list(list(name = "Charlottetown")))),
                   list(name = "Nova Scotia",
                   children = list(list(name = "Halifax"))",
                   list(name = "New Brunswick",
                   children = list(list(name = "Fredericton"))",
                   list(name = "Quebec",
                   children = list(list(name = "Montreal"),
                   list(name = "Quebec City"))",
                   list(name = "Ontario",
                   children = list(list(name = "Toronto"),
                   list(name = "Ottawa"))",
                   list(name = "Manitoba",
                   children = list(list(name = "Winnipeg"))",
                   list(name = "Saskatchewan",
                   children = list(list(name = "Regina"))",
                   list(name = "Nunavut",
                   children = list(list(name = "Iqaluit"))",
                   list(name = "NWT",
                   children = list(list(name = "Yellowknife"))",
                   list(name = "Alberta",
                   children = list(list(name = "Edmonton"))",
                   list(name = "British Columbia"),
```
forceNetwork

```r
children = list(list(name = "Victoria"),
               list(name = "Vancouver")),
list(name = "Yukon",
    children = list(list(name = "Whitehorse"))
)

diagonalNetwork(List = CanadaPC, fontSize = 10)

## End(Not run)
```

---

**forceNetwork**  
Create a D3 JavaScript force directed network graph.

**Description**  
Create a D3 JavaScript force directed network graph.

**Usage**  
```r
forceNetwork(Links, Nodes, Source, Target, Value, NodeID, Nodesize, Group,
             height = NULL, width = NULL,
             colourScale = JS("d3.scaleOrdinal(d3.schemeCategory20);"), fontSize = 7,
             fontFamily = "serif", linkDistance = 50,
             linkWidth = JS("function(d) { return Math.sqrt(d.value); }"),
             radiusCalculation = JS(" Math.sqrt(d.nodesize)+6"), charge = -30,
             linkColour = "#666", opacity = 0.6, zoom = FALSE, legend = FALSE,
             arrows = FALSE, bounded = FALSE, opacityNoHover = 0,
             clickAction = NULL)
```

**Arguments**  
- **Links** a data frame object with the links between the nodes. It should include the Source and Target for each link. These should be numbered starting from 0. An optional Value variable can be included to specify how close the nodes are to one another.
- **Nodes** a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the Source variable column in the Links data frame. Currently only a grouping variable is allowed.
- **Source** character string naming the network source variable in the Links data frame.
- **Target** character string naming the network target variable in the Links data frame.
- **Value** character string naming the variable in the Links data frame for how wide the links are.
- **NodeID** character string specifying the node IDs in the Nodes data frame.
- **Nodesize** character string specifying the a column in the Nodes data frame with some value to vary the node radius's with. See also radiusCalculation.
Group  character string specifying the group of each node in the Nodes data frame.
height  numeric height for the network graph’s frame area in pixels.
width  numeric width for the network graph’s frame area in pixels.
colourScale  character string specifying the categorical colour scale for the nodes. See https://github.com/d3/d3/blob/master/API.md#ordinal-scales.
fontSize  numeric font size in pixels for the node text labels.
fontFamily  font family for the node text labels.
linkDistance  numeric or character string. Either numeric fixed distance between the links in pixels (actually arbitrary relative to the diagram’s size). Or a JavaScript function, possibly to weight by Value. For example: linkDistance = JS("function(d){return d.value * 10}").
linkWidth  numeric or character string. Can be a numeric fixed width in pixels (arbitrary relative to the diagram’s size). Or a JavaScript function, possibly to weight by Value. The default is linkWidth = JS("function(d) { return Math.sqrt(d.value); }").
radiusCalculation  character string. A javascript mathematical expression, to weight the radius by Nodesize. The default value is radiusCalculation = JS("Math.sqrt(d.nodesize)+6").
charge  numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).
linkColour  character vector specifying the colour(s) you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
opacity  numeric value of the proportion opaque you would like the graph elements to be.
zoom  logical value to enable (TRUE) or disable (FALSE) zooming.
legend  logical value to enable node colour legends.
arrows  logical value to enable directional link arrows.
bounded  logical value to enable (TRUE) or disable (FALSE) the bounding box limiting the graph’s extent. See http://bl.ocks.org/mbostock/1129492.
opacityNoHover  numeric value of the opacity proportion for node labels text when the mouse is not hovering over them.
clickAction  character string with a JavaScript expression to evaluate when a node is clicked.

Source


See Also

JS.
Examples

# Load data
data(MisLinks)
data(MisNodes)
# Create graph
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4, zoom = TRUE)

# Create graph with legend and varying node radius
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Nodesize = "size",
            radiusCalculation = "Math.sqrt(d.nodesize)+6",
            Group = "group", opacity = 0.4, legend = TRUE)

# Create graph directed arrows
forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4, arrows = TRUE)

## Not run:
#### JSON Data Example
# Load data JSON formated data into two R data frames
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://cdn.rawgit.com/christophergandrud/networkD3/",
              "master/JSONdata/miserables.json")
MisJson <- jsonlite::fromJSON(URL)
# Create graph
forceNetwork(Links = MisJson$links, Nodes = MisJson$nodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4)

# Create graph with zooming
forceNetwork(Links = MisJson$links, Nodes = MisJson$nodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4, zoom = TRUE)

# Create a bounded graph
forceNetwork(Links = MisJson$links, Nodes = MisJson$nodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4, bounded = TRUE)

# Create graph with node text faintly visible when no hovering
forceNetwork(Links = MisJson$links, Nodes = MisJson$nodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.4, bounded = TRUE,
            opacityNoHover = TRUE)
## Specify colours for specific edges

# Find links to Valjean (11)
which(MisNodes == "Valjean", arr = TRUE)[1] - 1
ValjeanInds = which(MisLinks == 11, arr = TRUE)[, 1]

# Create a colour vector
ValjeanCols = ifelse(1:nrow(MisLinks) %in% ValjeanInds, "#bf3eff", "#666")

forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 0.8, linkColour = ValjeanCols)

## Create graph with alert pop-up when a node is clicked. You're
# unlikely to want to do exactly this, but you might use
# Shiny.onInputChange() to allocate d.XXX to an element of input
# for use in a Shiny app.

MyClickScript <- "alert("You clicked " + d.name + " which is in row " +
               (d.index + 1) + " of your original R data frame");"

forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
            Target = "target", Value = "value", NodeID = "name",
            Group = "group", opacity = 1, zoom = FALSE,
            bounded = TRUE, clickAction = MyClickScript)

## End(Not run)

The `igraph_to_networkD3` function is used to convert an igraph graph to a list suitable for networkD3. The function takes three arguments:

- `g`: an igraph class graph object.
- `group`: an object that contains node group values, for example, those created with igraph’s `membership` function.
- `what`: a character string specifying what to return. If `what = 'links'` or `what = 'nodes'` only the links or nodes are returned as data frames, respectively. If `what = 'both'` then both data frames will be return in a list.

The function is particularly useful when you want to visualize a graph using networkD3 and interact with the nodes by displaying pop-up messages.
Value

A list of link and node data frames or only the link or node data frames.

Examples

```r
# Load igraph
library(igraph)

# Use igraph to make the graph and find membership
karate <- make_graph("Zachary")
w <- cluster_walktrap(karate)
members <- membership(w)

# Convert to object suitable for networkD3
karate_d3 <- igraph_to_networkD3(karate, group = members)

# Create force directed network plot
forceNetwork(Links = karate_d3$links, Nodes = karate_d3$nodes,
            Source = 'source', Target = 'target', NodeID = 'name',
            Group = 'group')
```

```r
## Not run:
# Example with data from data frame
# Load data
## Original data from http://results.ref.ac.uk/DownloadSubmissions/ByUoa/21
## Load data
data("SchoolsJournals")

# Convert to igraph
SchoolsJournals <- graph.data.frame(SchoolsJournals, directed = FALSE)

# Remove duplicate edges
SchoolsJournals <- simplify(SchoolsJournals)

# Find group membership
wt <- cluster_walktrap(SchoolsJournals, steps = 6)
members <- membership(wt)

# Convert igraph to list for networkD3
sj_list <- igraph_to_networkD3(SchoolsJournals, group = members)

# Plot as a forceDirected Network
forceNetwork(Links = sj_list$links, Nodes = sj_list$nodes, Source = 'source',
            Target = 'target', NodeID = 'name', Group = 'group',
            zoom = TRUE, linkDistance = 200)
```

## End(Not run)
Description

Create character strings that will be evaluated as JavaScript

Usage

JS(...)

Arguments

... character string to evaluate

Source


| MisLinks | Les Miserables character links |

Description

A data file of links from Knuth’s Les Miserables characters data base.

Usage

MisLinks

Format

A data set with 254 observations of 3 variables.

Source

See Mike Bostock http://bl.ocks.org/mbostock/4062045.
**MisNodes**

*Les Miserables character nodes*

**Description**

A data file of nodes from Knuth’s Les Miserables characters data base.

**Usage**

MisNodes

**Format**

A data set with 77 observations of 2 variables, plus made up node size variable.

**Source**

See Mike Bostock [http://bl.ocks.org/mbostock/4062045](http://bl.ocks.org/mbostock/4062045).

---

**radialNetwork**  
*Create Reingold-Tilford Tree network diagrams.*

**Description**

Create Reingold-Tilford Tree network diagrams.

**Usage**

radialNetwork(List, height = NULL, width = NULL, fontSize = 10, 
fontFamily = "serif", linkColour = "#ccc", nodeColour = "#fff", 
nodeStroke = "steelblue", textColour = "#111", opacity = 0.9, 
margin = NULL)

**Arguments**

- **List**  
a hierarchical list object with a root node and children.
- **height**  
height for the network graph’s frame area in pixels (if NULL then height is automatically determined based on context)
- **width**  
numeric width for the network graph’s frame area in pixels (if NULL then width is automatically determined based on context)
- **fontSize**  
numeric font size in pixels for the node text labels.
- **fontFamily**  
font family for the node text labels.
- **linkColour**  
character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
nodeColour  character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).

nodeStroke  character string specifying the colour you want the node perimeter to be. Multiple formats supported (e.g. hexadecimal).

textColour  character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).

opacity  numeric value of the proportion opaque you would like the graph elements to be.

margin  an integer or a named list/vector of integers for the plot margins. If using a named list/vector, the positions top, right, bottom, left are valid. If a single integer is provided, then the value will be assigned to the right margin. Set the margin appropriately to accommodate long text labels.

Source


Mike Bostock: http://bl.ocks.org/mbostock/4063550.

Examples

## Not run:

#### Create tree from JSON formatted data
## Download JSON data
# Create URL. paste0 used purely to keep within line width.
URL <- paste0("https://cdn.rawgit.com/christophergandrud/networkD3/",
"master/JSONdata//flare.json")

## Convert to list format
Flare <- jsonlite::fromJSON(URL, simplifyDataFrame = FALSE)

## Recreate Bostock example from http://bl.ocks.org/mbostock/4063550
radialNetwork(List = Flare, fontSize = 10, opacity = 0.9)

#### Create a tree dendrogram from an R hclust object
hc <- hclust(dist(USArrests), "ave")
radialNetwork(as.radialNetwork(hc))
radialNetwork(as.radialNetwork(hc), fontFamily = "cursive")

#### Create tree from a hierarchical R list
For an alternative structure see: http://stackoverflow.com/a/30747323/1705044
CanadaPC <- list(name = "Canada", children = list(list(name = "Newfoundland",
    children = list(list(name = "St. John's"))),
    list(name = "PEI",
    children = list(list(name = "Charlottetown"))),
    list(name = "Nova Scotia",
    children = list(list(name = "Halifax"))),
    list(name = "New Brunswick",
    children = list(list(name = "Fredericton"))),
    list(name = "Quebec"),
...
Create a D3 JavaScript Sankey diagram

**Description**
Create a D3 JavaScript Sankey diagram

**Usage**

```r
sankeyNetwork(Links, Nodes, Source, Target, Value, NodeID, NodeGroup = NodeID, 
LinkGroup = NULL, units = "", 
colourScale = JS("d3.scaleOrdinal(d3.schemeCategory20);"), fontSize = 7, 
fontFamily = NULL, nodeWidth = 15, nodePadding = 10, margin = NULL, 
height = NULL, width = NULL, iterations = 32, sinksRight = TRUE)
```

**Arguments**

- **Links**: a data frame object with the links between the nodes. It should have include the Source and Target for each link. An optional Value variable can be included to specify how close the nodes are to one another.

- **Nodes**: a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the Source variable column in the Links data frame. Currently only grouping variable is allowed.

```r
children = list(list(name = "Montreal"), 
    list(name = "Quebec City"))),
list(name = "Ontario", 
    children = list(list(name = "Toronto"), 
    list(name = "Ottawa"))),
list(name = "Manitoba", 
    children = list(list(name = "Winnipeg"))),
list(name = "Saskatchewan", 
    children = list(list(name = "Regina"))),
list(name = "Nunavut", 
    children = list(list(name = "Iqaluit"))),
list(name = "NWT", 
    children = list(list(name = "Yellowknife"))),
list(name = "Alberta", 
    children = list(list(name = "Edmonton"))),
list(name = "British Columbia", 
    children = list(list(name = "Victoria"), 
    list(name = "Vancouver"))),
list(name = "Yukon", 
    children = list(list(name = "Whitehorse")))
)
```

```r
radialNetwork(List = CanadaPC, fontSize = 10)
```

## End(Not run)
Source character string naming the network source variable in the Links data frame.
Target character string naming the network target variable in the Links data frame.
Value character string naming the variable in the Links data frame for how far away the nodes are from one another.
NodeID character string specifying the node IDs in the Nodes data frame. Must be 0-indexed.
NodeGroup character string specifying the node groups in the Nodes. Used to color the nodes in the network.
LinkGroup character string specifying the groups in the Links. Used to color the links in the network.
units character string describing physical units (if any) for Value
colourScale character string specifying the categorical colour scale for the nodes. See https://github.com/d3/d3/blob/master/API.md#ordinal-scales.
fontSize numeric font size in pixels for the node text labels.
fontFamily font family for the node text labels.
odeWidth numeric width of each node.
nodePadding numeric essentially influences the width height.
margin an integer or a named list/vector of integers for the plot margins. If using a named list/vector, the positions top, right, bottom, left are valid. If a single integer is provided, then the value will be assigned to the right margin. Set the margin appropriately to accomodate long text labels.
height numeric height for the network graph’s frame area in pixels.
width numeric width for the network graph’s frame area in pixels.
iterations numeric. Number of iterations in the diagramm layout for computation of the depth (y-position) of each node. Note: this runs in the browser on the client so don’t push it too high.
sinksRight boolean. If TRUE, the last nodes are moved to the right border of the plot.

Source

D3.js was created by Michael Bostock. See http://d3js.org/ and, more specifically for Sankey diagrams http://bost.ocks.org/mike/sankey/.

See Also

JS

Examples

## Not run:
# Recreate Bostock Sankey diagram: http://bost.ocks.org/mike/sankey/
# Load energy projection data
URL <- paste0('https://cdn.rawgit.com/christophergandrud/networkD3/',
               'master/JSONdata/energy.json')
energy <- jsonlite::fromJSON(URL)
saveNetwork

# Plot
sankeyNetwork(Links = energy$links, Nodes = energy$nodes, Source = 'source',
              Target = 'target', Value = 'value', NodeID = 'name',
              units = 'TWh', fontSize = 12, nodeWidth = 30)

# Colour links
energy$links$energy_type <- sub('.*','
    energy$nodes[energy$links$source + 1, 'name'])

sankeyNetwork(Links = energy$links, Nodes = energy$nodes, Source = 'source',
              Target = 'target', Value = 'value', NodeID = 'name',
              LinkGroup = 'energy_type', NodeGroup = NULL)

## End(Not run)

saveNetwork(network, file, selfcontained = TRUE)

Arguments

- **network**: Network to save (e.g. result of calling the function simpleNetwork).
- **file**: File to save HTML into
- **selfcontained**: Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory.

SchoolsJournals

Edge list of REF (2014) journal submissions for Politics and International Relations

Description

Edge list of REF (2014) journal submissions for Politics and International Relations
simpleNetwork

**Usage**

`SchoolsJournals`

**Format**

A data set with 2732 rows and 3 variables.

**Source**

See REF 2014 [http://results.ref.ac.uk/DownloadSubmissions/ByUoa/21](http://results.ref.ac.uk/DownloadSubmissions/ByUoa/21).

---

**simpleNetwork**

*Function for creating simple D3 JavaScript force directed network graphs.*

**Description**

simpleNetwork creates simple D3 JavaScript force directed network graphs.

**Usage**

```r
simpleNetwork(Data, Source = 1, Target = 2, height = NULL, width = NULL, linkDistance = 50, charge = -30, fontSize = 7, fontFamily = "serif", linkColour = "#666", nodeColour = ",#3182bd", opacity = 0.6, zoom = F)
```

**Arguments**

- **Data**
  a data frame object with three columns. The first two are the names of the linked units. The third records an edge value. (Currently the third column doesn’t affect the graph.)

- **Source**
  character string naming the network source variable in the data frame. If Source = NULL then the first column of the data frame is treated as the source.

- **Target**
  character string naming the network target variable in the data frame. If Target = NULL then the second column of the data frame is treated as the target.

- **height**
  height for the network graph’s frame area in pixels (if NULL then height is automatically determined based on context)

- **width**
  numeric width for the network graph’s frame area in pixels (if NULL then width is automatically determined based on context)

- **linkDistance**
  numeric distance between the links in pixels (actually arbitrary relative to the diagram’s size).

- **charge**
  numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).

- **fontSize**
  numeric font size in pixels for the node text labels.

- **fontFamily**
  font family for the node text labels.
simpleNetwork

linkColour character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).

nodeColour character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).

opacity numeric value of the proportion opaque you would like the graph elements to be.

zoom logical value to enable (TRUE) or disable (FALSE) zooming.

Source

D3.js was created by Michael Bostock. See http://d3js.org/ and, more specifically for directed networks https://github.com/d3/d3/blob/master/API.md#forces-d3-force.

Examples

# Fake data
NetworkData <- data.frame(Source, Target)

# Create graph
simpleNetwork(NetworkData)
simpleNetwork(NetworkData, fontFamily = "sans-serif")
Index

* datasets
  MisLinks, 14
  MisNodes, 15
  SchoolsJournals, 19

as.radialNetwork, 2
chordNetwork, 3
chordNetworkOutput, 4
dendroNetwork, 5
dendroNetworkOutput (chordNetworkOutput), 4
diagonalNetwork, 7
diagonalNetworkOutput (chordNetworkOutput), 4
forceNetwork, 9
forceNetworkOutput (chordNetworkOutput), 4

htmlwidgets, 14
igraph_to_networkD3, 12
JS, 10, 13, 18

membership, 12
MisLinks, 14
MisNodes, 15

networkD3 (networkD3-package), 2
networkD3-package, 2
networkD3-shiny (chordNetworkOutput), 4

radialNetwork, 15
radialNetworkOutput (chordNetworkOutput), 4
renderChordNetwork (chordNetworkOutput), 4
renderDendroNetwork (chordNetworkOutput), 4
renderDiagonalNetwork (chordNetworkOutput), 4
renderForceNetwork (chordNetworkOutput), 4
renderRadialNetwork (chordNetworkOutput), 4
renderSankeyNetwork (chordNetworkOutput), 4
renderSimpleNetwork (chordNetworkOutput), 4
sankeyNetwork, 17
sankeyNetworkOutput (chordNetworkOutput), 4
saveNetwork, 19
SchoolsJournals, 19
simpleNetwork, 20
simpleNetworkOutput (chordNetworkOutput), 4