Package ‘RGraph2js’

March 28, 2024

Type Package

Title Convert a Graph into a D3js Script

Version 1.30.0

Date 2016-05-09

Imports utils, whisker, rjson, digest, graph

Suggests RUnit, BiocStyle, BiocGenerics, xtable, sna

Description Generator of web pages which display interactive network/graph visualizations with D3js, jQuery and Raphael.

License GPL-2

SystemRequirements jQuery, jQueryUI, qTip2, D3js and Raphael are required Javascript libraries made available via the online CDNJS service (http://cdnjs.cloudflare.com).

Collate 'RGraph2js-package.R' 'dataforming.R' 'utils.R' 'jstemplate.R' 'htmltemplate.R' 'graph2js.R'

biocViews Visualization, Network, GraphAndNetwork, ThirdPartyClient

NeedsCompilation no

Author Stephane Cano [aut, cre], Sylvain Gubian [aut], Florian Martin [aut]

Maintainer Stephane Cano <DL.RSupport@pmi.com>

git_url https://git.bioconductor.org/packages/RGraph2js

git_branch RELEASE_3_18

git_last_commit 361a0e9

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-03-27
R topics documented:

generateOptionsJSCode ................................................ 2
getAdjMat ................................................................. 3
getDefaultOptions ......................................................... 3
getDefaultToolParameters ............................................... 7
getEdgesDataFrame ....................................................... 8
getHTMLContainerCode ................................................... 9
getHTMLStyleCode ......................................................... 9
getJSCode ................................................................. 10
getNodesDataFrame ....................................................... 10
getUUID ................................................................. 11
graph2js ................................................................. 12
graph2json ............................................................... 14

Index


generateOptionsJSCode  Generate javascript code based on general options, options for containers and network data

Description

Generate javascript code based on general options, options for containers and network data

Usage

generateOptionsJSCode(opts)

Arguments

opts  list containing general options for GraphRender component

Value

String containing JS code for component options

Author(s)

Sylvain Gubian <DL.RSupport@pmi.com>
getAdjMat

Get a RGraph2js compatible adjacency matrix from the provided R object.

Usage

getAdjMat(A)

Arguments

A  signed weighted adjacency matrix or an instance of the class graphAM, graphBAM, graphNEL or clusterGraph from the graph package

Value

the RGraph2js compatible adjacency matrix

Author(s)

Stephane Cano <stephane.cano@pmi.com>, PMP SA.

defaultOptions

Generate a list containing parameters for the D3js component’s options with default values

Description

Generate a list containing parameters for the D3js component’s options with default values

Usage

defaultOptions()
Description of the available options

\[ w \] : width of the component in pixels
\[ h \] : height of the component in pixels

\[ \text{minZoomFactor} \] : float \([0, n]\), 1 means 100%
\[ \text{maxZoomFactor} \] : float \([0, n]\), 1 means 100%

\[ \text{layout-forceLinkDistance} \] : float
If distance is specified, sets the target distance between linked nodes to the specified value. If distance is not specified, returns the layout’s current link distance, which defaults to 20. Typically, the distance is specified in pixels; however, the units are arbitrary relative to the layout’s size.

\[ \text{layout-forceCharge} \] : float
If charge is specified, sets the charge strength to the specified value. If charge is not specified, returns the current charge strength, which defaults to -900. A negative value results in node repulsion, while a positive value results in node attraction. For graph layout, negative values should be used; for n-body simulation, positive values can be used. All nodes are assumed to be infinitesimal points with equal charge and mass. Charge forces are implemented efficiently via the Barnes-Hut algorithm, computing a quadtree for each tick. Setting the charge force to zero disables computation of the quadtree, which can noticeably improve performance if you do not need n-body forces.

\[ \text{layout-linkStrength} \] : float \([0, 1]\)
If strength is specified, sets the strength (rigidity) of links to the specified value in the range \([0, 1]\). If strength is not specified, returns the layout’s current link strength, which defaults to 1.

\[ \text{layout-friction} \] : float
If friction is specified, sets the friction coefficient to the specified value. If friction is not specified, returns the current coefficient, which defaults to 0.9. The name of this parameter is perhaps misleading; it does not correspond to a standard physical coefficient of friction. Instead, it more closely approximates velocity decay: at each tick of the simulation, the particle velocity is scaled by the specified friction. Thus, a value of 1 corresponds to a frictionless environment, while a value of 0 freezes all particles in place. Values outside the range \([0, 1]\) are not recommended and may have destabilizing effects.

\[ \text{layout-chargeDistance} \] : float
If distance is specified, sets the maximum distance over which charge forces are applied. If distance is not specified, returns the current maximum charge distance, which defaults to infinity. Specifying a finite charge distance improves the performance of the force layout and produces a more localized layout; distance-limited charge forces are especially useful in conjunction with custom gravity.

\[ \text{layout-theta} \] : float
If theta is specified, sets the Barnes-Hut approximation criterion to the specified value. If theta is not specified, returns the current value, which defaults to 0.8. Unlike links, which only affect two linked nodes, the charge force is global: every node affects every other node, even if they are on disconnected subgraphs. To avoid quadratic performance slowdown for large graphs, the force layout uses the Barnes-Hut approximation which takes \(O(n \log n)\) per tick. For each tick, a quadtree is created to store the current node positions; then for each node, the sum charge force of all other nodes on the given node are computed. For clusters of nodes that are far away, the charge force is
getDefaultsOptions

approximated by treating the distance cluster of nodes as a single, larger node. Theta determines
the accuracy of the computation: if the ratio of the area of a quadrant in the quadtree to the distance
between a node to the quadrant’s center of mass is less than theta, all nodes in the given quadrant
are treated as a single, larger node rather than computed individually.

layout_gravity : float
If gravity is specified, sets the gravitational strength to the specified value. If gravity is not speci-
fied, returns the current gravitational strength, which defaults to 0.1. The name of this parameter
is perhaps misleading; it does not correspond to physical gravity (which can be simulated using a
positive charge parameter). Instead, gravity is implemented as a weak geometric constraint simi-
lar to a virtual spring connecting each node to the center of the layout’s size. This approach has
nice properties: near the center of the layout, the gravitational strength is almost zero, avoiding any
local distortion of the layout; as nodes get pushed farther away from the center, the gravitational
strength becomes strong in linear proportion to the distance. Thus, gravity will always overcome
repulsive charge forces at some threshold, preventing disconnected nodes from escaping the layout.
Gravity can be disabled by setting the gravitational strength to zero. If you disable gravity, it is
recommended that you implement some other geometric constraint to prevent nodes from escaping
the layout, such as constraining them within the layout’s bounds.

maxLayoutIterations : the max allowed number to perform

displayNetworkEveryNLayoutIterations : 1 means always, 0 to display only on layout comple-
tion

optimizeDisplayWhenLayoutRunning : boolean, TRUE to simplify the display when the layout
engine is running
FALSE otherwise.

nodeSize : size of the node in pixels

nodeRoundedCornerPixels : apply rounded corners on rectangle like shapes

displayNodeLabels : boolean, display node names besides them

nodeBorderColor : RGB hex color

leadingNodeBorderColor : RGB hex color

noneLeadingNodeOpacity : float [0,1], 1 means fully opaque

nodeLabelsColor : RGB hex color, example "#444444"

nodeLabelsFont : example "6px sans-serif"

dragNodeBorderColor : the node border color to apply on dragging

selectNodeBorderColor : the node border color to apply on left-click, "#ff0000"
displayBarPlotsInsideNodes : boolean, display barplots inside nodes

barplotInNodeTooltips : boolean, display barplots inside node’s tooltips

barplotInsideNodeBorderColor : the barplot borders color, example ‘#000000’
barplotInsideNodeBorderWidth : the barplot borders width in pixels, example ‘2px’

nodeTooltipOpacity : float [0,1], 1 means fully opaque (for link tooltips as well)

displayBarplotTooltips : boolean, (dis/en)able tooltips for each barplot’s bar

nodeTooltipActivationDelay : milliseconds (for link tooltips as well)
nodeTooltipDeactivationDelay : milliseconds (for link tooltips as well)

barplotInNodeTooltipsFontSize : pixels

enableNodeDragging : boolean, allow/deny node dragging

jsFunctionToCallOnNodeClick : name of the javascript function to call on node click
example:
To call the following function
var myfunction = function(nodeObj) alert(nodeObj.name); ;
you should set jsFunctionToCallOnNodeClick='myfunction'

displayColorScale : show a color scale in the toolbar

scaleGradient : define the linear color gradient
Linear gradient format is "<angle>-<colour>[<-colour>[<-offset>][,<offset>]]*-<colour>"
examples: "90-#fff-#000" => 90 degree gradient from white to black
"0-#fff-#f00:20-#000" -> 0 degree gradient from white via red (at 20%) to black.

scaleLabelsFontFamily : example "monospace"

scaleLabelsFontSize : in pixels

scaleHeight : in pixels

scaleTickSize : in pixels

scaleTicksPercents : to draw a tick every 20%: "[20,40,60,80,100]"

exportCGI : boolean, enable a CGI conversion in the export function, permit only the SVG export otherwise
getDefaultToolParameters

Function which generates a list containing parameters for Tools in the D3js component with default values

Description

Function which generates a list containing parameters for Tools in the D3js component with default values

Usage

getDefaultToolParameters()

Value

list of parameters with default values

Author(s)

Sylvain Gubian <DL.RSupport@pmi.com>
getEdgesDataFrame

Create Edges data.frame from Adjacency matrix and properties

Description

Create Edges data.frame from Adjacency matrix and properties

Usage

getEdgesDataFrame(A, eGlobal = NULL, eProp = NULL)

Arguments

A  signed weighted adjacency matrix

eGlobal  A list of properties for assigning all edges. Default value is NULL

eProp  A data.frame for assigning some nodes properties. Default value is NULL

Value

A data.frame

Author(s)

Sylvain Gubian <DL.RSupport@pmi.com>

Examples

v <- c(0, 0, 1, 1, 0,
      0, 0, 0, 0, 0,
      -1, 0, 0, 1, 0)
a <- matrix(v, 3, 5)
colnames(a) <- LETTERS[1:5]
rownames(a) <- LETTERS[1:3]
eGlobal <- list(color="#5555ff")
eProp <- data.frame(from=c('A','C'), to=c('B', 'A'), width=c(2,2))
getEdgesDataFrame(A=a, eGlobal=eGlobal, eProp=eProp)
getHTMLContainerCode

Generate a HTML table node code for component based on template

Description
Generate a HTML table node code for component based on template

Usage
getHTMLContainerCode(id, toolParam)

Arguments
id String for component identification
toolParam list containing options for tools options of the GraphRender component

Value
String which is the HTML code generated

Author(s)
Sylvain Gubian <DL.RSupport@pmi.com>

getHTMLStyleCode
Generate a HTML style code for component based on template

Description
Generate a HTML style code for component based on template

Usage
getHTMLStyleCode(id)

Arguments
id String for component identification

Value
String which is the HTML style code generated

Author(s)
Sylvain Gubian <DL.RSupport@pmi.com>
**getJSCode**

*Generate javascript code based on general options and network data*

**Description**

Generate javascript code based on general options and network data

**Usage**

```r
getJSCode(dataJson, id, opts, toolParam)
```

**Arguments**

- **dataJson** list containing network data for nodes and links
- **id** String for component identification
- **opts** list containing general options for GraphRender component
- **toolParam** list containing urls to jquery, jquery-ui, d3js, GraphRender JS library, options for the GraphRender tool

**Value**

String corresponding to JS code

**Author(s)**

Sylvain Gubian <DL.RSupport@pmi.com>

---

**getNodesDataFrame**

*Create Nodes data.frame from Adjacency matrix and properties for specific nodes*

**Description**

Create Nodes data.frame from Adjacency matrix and properties for specific nodes

**Usage**

```r
getNodesDataFrame(A, nGlobal = NULL, nProp = NULL)
```

**Arguments**

- **A** signed weighted adjacency matrix
- **nGlobal** A list of properties for assigning all nodes. Default value is NULL
- **nProp** A data.frame for assigning some nodes properties Default value is NULL
getUUID

Function which generates a UUID version 4

Description

Function which generates a UUID version 4

Usage

getUUID(seed = NULL)

Arguments

seed Integer for seeding the R random generator

Value

String corresponding to the UUID generated.

Examples

v <- c(0, 0, 1, 1, 0,
       0, 0, 0, 0, 0,
       -1, 0, 0, 1, 0)
a <- matrix(v, 3, 5)
colnames(a) <- LETTERS[1:5]
rownames(a) <- LETTERS[1:3]
nGlobal <- list(color="#dedeff")
nProp <- data.frame(shape=c('triangle', 'lozenge'))
rownames(nProp) <- c('C', 'E')
getNodesDataFrame(A=a, nGlobal=nGlobal, nProp=nProp)
graph2js

Generate the JSON code using D3js that draws a network from Adjacency matrix and edges, nodes properties.

Description

Generate the JSON code using D3js that draws a network from Adjacency matrix and edges, nodes properties.

Usage

```r
graph2js(A, innerValues = NULL, innerColors = NULL, innerTexts = NULL,
starplotValues = NULL, starplotColors = NULL, starplotLabels = NULL,
starplotTooltips = NULL, starplotUrlLinks = NULL,
starplotSectorStartedRad = NULL, starplotCircleFillColor = NULL,
starplotCircleFillOpacity = NULL, nodesGlobal = NULL, nodesProp = NULL,
edgesGlobal = NULL, edgesProp = NULL, outputDir = NULL,
filename = NULL, opts = list(), userCssStyles = NULL,
toolsPar = list(), id = getUUID())
```

Arguments

- `A` signed weighted adjacency matrix or an instance of the class `graphAM`, `graphBAM`, `graphNEL` or `clusterGraph` from the `graph` package
- `innerValues` A matrix of inner node values to display Barplot or other component. In a row, numerical values for a node.
- `innerColors` A matrix of colors for coloring the inner node barplot or component. In a row, colors values for a node.
- `innerTexts` A matrix of labels for each bar in inner barplots. In a row, labels values for a node.
- `starplotValues` A matrix of [0,1] values for starpot sectors size
- `starplotColors` A matrix of hex RGB colors for sectors colors
- `starplotLabels` A matrix of labels identifying the sectors
- `starplotTooltips` A matrix of text or even html content for the sectors tooltips
- `starplotUrlLinks` A matrix of text for the sectors url links
- `starplotSectorStartedRad` A matrix with a single column of [0,2PI] values for the sector start in radians
- `starplotCircleFillColor` A matrix of hex RGB colors for the circle background
- `starplotCircleFillOpacity` A matrix of [0.0,1.0] values for the background opacity
- `nodesGlobal` A list of global nodes properties.
nodesProp
A data.frame object containing properties for specific nodes width, shape (in 'rect', 'circle', 'lozenge', 'triangle'), link, tooltip, highlight.X (X from 0 to N for animation) columns

edgesGlobal
A list of global edges properties.

edgesProp
A data.frame object containing properties for specific edges from, to, width, type, link, color columns

outputDir
String that corresponds to the path to a folder or file where js code and dependencies will be generated. If NULL is provided, javascript code is returned in the returned list by the function with the slots:
'jsIncludes' A character string containing JS code for including the necessary JS files
'styling' A character string which contains the CSS code for the GraphRenderer component
'js' A character string containing the JavaScript code for the rendering of the data
'html' A character string containing the HTML code for the rendering of the component

filename
String the name of the result HTML file, a name will be automatically generated if not provided and by default.

opts
list of options of the GraphRenderer component (See getDefaultOptions function available options)

userCssStyles
String containing user css styles. (See starplot demo)

toolsPar
list of options for tools attached to GraphRenderer component. (See getDefaultToolParameters for details)

id
function, Unique IDs generator, Internal function getUUID by default.

Value
A list containing information of the generated js code.

Examples
v <- c(0, 0, 1, 1, 0,
0, 0, 0, 0, 0,
-1, 0, 0, 1, 0)
a <- matrix(v, 3, 5)
colnames(a) <- LETTERS[1:5]
rownames(a) <- LETTERS[1:3]
g <- graph2js(a)
graph2json

Generates JSON string corresponding the the graph description

Description

Generates JSON string corresponding the the graph description

Usage

    graph2json(ndf, edf, innerValues = NULL, innerColors = NULL,
               innerTexts = NULL, starplotColors = NULL, starplotValues = NULL,
               starplotLabels = NULL, starplotTooltips = NULL, starplotUrlLinks = NULL,
               starplotSectorStartRad = NULL, starplotCircleFillColor = NULL,
               starplotCircleFillOpacity = NULL)

Arguments

    ndf       A data.frame corresponding to nodes definition
    edf       A data.frame corresponding to edges definition
    innerValues       A matrix of numerical values for plotting in the node
    innerColors       A matrix of string colors values for plotting in the node
    innerTexts       A matrix of strings for plotting in the node
    starplotColors       A matrix of hex RGB colors for sectors colors
    starplotValues       A matrix of [0,1] values for starpot sectors size
    starplotLabels       A matrix of labels identifying the sectors
    starplotTooltips       A matrix of text or even html content for the sectors tooltips
    starplotUrlLinks       A matrix of text for the sectors url links
    starplotSectorStartRad       A matrix with a single column of [0,2PI] values for the sector start in radians
    starplotCircleFillColor       A matrix of hex RGB colors for the circle background
    starplotCircleFillOpacity       A matrix of [0.0,1.0] values for the circle background opacity

Value

    A JSON string with formatting

Author(s)

    Sylvain Gubian <DL.RSupport@pmi.com>
Examples

v <- c(0, 0, 1, 1, 0,
      0, 0, 0, 0, 0,
      -1, 0, 0, 1, 0)
a <- matrix(v, 3, 5)
colnames(a) <- LETTERS[1:5]
rownames(a) <- LETTERS[1:3]

nGlobal <- list(color="#dedeff")
nProp <- data.frame(shape=c('triangle', 'lozenge'))
rownames(nProp) <- c('C', 'E')
ndf <- getNodesDataFrame(A=a, nGlobal=nGlobal, nProp=nProp)

eGlobal <- list(color="#5555ff")
eProp <- data.frame(from=c('A', 'C'), to=c('B', 'A'), width=c(2, 2))
edf <- getEdgesDataFrame(A=a, eGlobal=eGlobal, eProp=eProp)

graph2json(ndf=ndf, edf=edf)
Index

generateOptionsJSCode, 2
getAdjMat, 3
getDefaultOptions, 3
getDefaultToolParameters, 7
getEdgesDataFrame, 8
getHTMLContainerCode, 9
getHTMLStyleCode, 9
getJSCode, 10
getNodesDataFrame, 10
getUUID, 11
graph2js, 12
graph2json, 14