Package 'riverplot'

October 14, 2022

Type Package

Title Sankey or Ribbon Plots

Version 0.10

Date 2021-01-22

Author January Weiner <january.weiner@gmail.com>

Maintainer January Weiner <january.weiner@gmail.com>

Description Sankey plots are a type of diagram that is convenient to illustrate how flow of information, resources etc. separates and joins, much like observing how rivers split and merge. For example, they can be used to compare different clusterings.

URL https://logfc.wordpress.com

License GPL (>= 2.0)

Encoding UTF-8

Imports methods,RColorBrewer

LazyData true

Suggests knitr,maptools,qpdf,rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-01-22 14:10:02 UTC

R topics documented:

`riverplot-package` *Sankey / ribbon diagrams*

Description

Sankey / ribbon diagrams

Details

Sankey diagrams are a type of flow diagrams, in which the width of the arrows is proportional to the quantity they illustrate. Riverplot allows the creation, in R, of a basic type of Sankey diagrams.

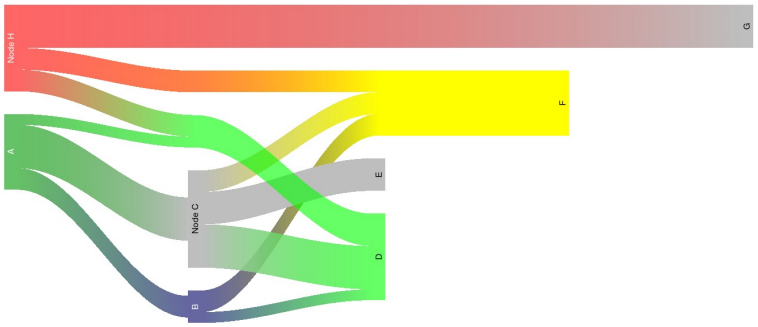
First, you need to create a specific riverplot object that can be directly plotted. (Use `riverplot.example` to generate an example object).

The simplest way is to create a graph-like representation of you diagram as a list of nodes; each item in the list is a list of partner nodes. Furthermore, you need to know at which position (from left to right) each node resides. Please take a look at the example section in the `makeRiver` function.

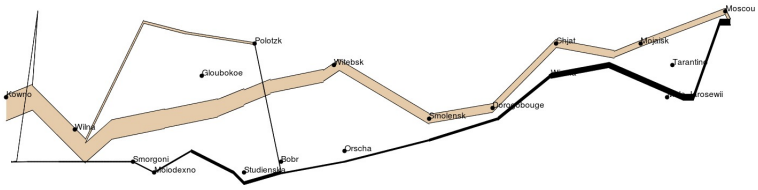
Once you have created a riverplot object with one of the above methods (or manually), you can plot it either with `plot(x)` or `riverplot(x)` (see `riverplot` for details).

Mini-gallery

Simple example from `riverplot.example` function: `plot( riverplot.example() )`.



Recreation of the famous figure by Charles Minard (see `minard` for details).



Author(s)

January Weiner <january.weiner@gmail.com>

`bglabel` *Label with background*

Description

Create a label with background

Usage

```
bglabel(
  x,
  y,
  text,
  bg = "#cccccc99",
  margin = 0.5,
  border = NA,
  pos = "center",
  cex = 1,
  ...
)
```

Arguments

`x, y`	numeric vectors (coordinates)
`text`	a character vector of labels
`bg`	character vector; background color for the labels
`margin`	numeric vector; margin (in percentage of a single character) for width and height around the labels
`border`	character vector; see `rect` for details
`pos`	character vector; position where labels should be placed, relative to the coordinates. Can be one of "topleft", "top", "topright", "left", "center", "right", "bottomleft", "bottom" and "bottomright".
`cex`	numeric vector; cex to be used for drawing the text
`...`	any further parameters are passed to the `text` function

Details

Creates a label with a background, a little extra margin (if necessary) etc.

`colorRampPaletteAlpha` *Color interpolation*

Description

These functions are replacements for colorRamp and colorRampPalette from the package grDevices, the only difference being that they also interpolate the alpha channel (i.e. transparency).

Usage

```
colorRampPaletteAlpha(colors, ...)

colorRampAlpha(colors, bias = 1, interpolate = c("linear", "spline"))
```

Arguments

`colors`	colors to interpolate; must be a valid argument to `col2rgb()`.
`...`	arguments to pass to `colorRamp`.
`bias`	a positive number. Higher values give more widely spaced colors at the high end.
`interpolate`	use spline or linear interpolation

Details

These functions are replacements for `colorRamp` and `colorRampPalette` from the package grDevices. There are two differences: (i) these functions also interpolate the alpha channel (i.e. transparency) and (ii) there is no `space` parameter (only `rgb` space is allowed). For all the other details, see descriptions of the original package.

Value

Both functions return a function which takes an integer argument. For details, see description of `colorRampPalette`

Examples

```
colorRampPaletteAlpha( c( "#FF000033", "#00FF0099" ) )( 5 )
```

curveseg *Draw a curved segment*

Description

Draws a curved segment from point (x0,y0) to (x1,y1). The segment is a framgent of a sinusoid, has a defined width and can either have a single color or a color gradient.

Usage

```
curveseg(
  x0,
  x1,
  y0,
  y1,
  width = 1,
  nsteps = 50,
  col = "#ffcc0066",
  grad = NULL,
  lty = 1,
  form = c("sin", "line"),
  fix.pdf = 0
)
```

Arguments

x0	X coordinate of the starting point
x1	X coordinate of the end point
y0	X coordinate of the starting point
y1	X coordinate of the end point
width	Width of the segment to plot
nsteps	Number of polygons to use for the segments. The more, the smoother the picture, but at the same time, the more time-consuming to display.
col	Color to use. Ignored if grad is not NULL.
grad	Gradient to use. Can be anything that colorRampPalette can understand.
lty	Line type for drawing of the segment. Use lty=0 for no line.
form	"sin" for a sinusoidal segment. "line" for a straight segment.
fix.pdf	Draw a border around segments with line type lty in a desperate attempt to fix the PDF output.

Value

no value is returned

Examples

```
# a DNA strand
plot.new()
par( usr= c( 0, 4, -2.5, 2.5 ) )

w    <- 0.4
cols <- c( "blue", "green" )
init <- c( -0.8, -0.5 )
pos  <- c( 1, -1 )
step <- 0.5

for( i in rep( rep( c( 1, 2 ), each= 2 ), 5 ) ) {
  curveseg( init[i], init[i] + step, pos[1], pos[2], width= w, col= cols[i] )
  init[i] <- init[i] + step
  pos <- pos * -1
}
```

`makeRiver` *Create a new riverplot object*

Description

Create a new riverplot object

Usage

```
makeRiver(
  nodes,
  edges,
  node_labels = NULL,
  node_xpos = NULL,
  node_ypos = NULL,
  styles = NULL,
  node_styles = NULL,
  edge_styles = NULL,
  default_style = NULL
)
```

Arguments

`nodes`	Data frame with node ID's, positions and optionally other information
`edges`	A named list or a data frame specifying the edges between the nodes.
`node_labels`	A named character vector of labels for the nodes
`node_xpos`	A named vector of numeric values specifying the horizontal positions on the plot.
`node_ypos`	A named vector of numeric values specifying the vertical positions on the plot.

styles	A named list specifying the styles for the nodes and edges
node_styles	Deprecated
edge_styles	Deprecated
default_style	list containing style information which is applied to every node and every edge

Details

Functions to create a new object of the riverplot class from the provided data.

makeRiver creates a plot from an object which specifies the graph directly, i.e. all nodes, their horizontal positions on the plot, provided styles etc. See sections below for detailed explanations.

Value

A riverplot object which can directly be plotted.

Structure of the riverplot objects

A riverplot object is a list with the following entries:

nodes A data frame specifying the nodes, containing at least the columns "ID" and "x" (horizontal position of the node). Optionally, it can also contain columns "labels" (the labels to display) and "y" (vertical position of the node on the plot)

edges A data frame specifying the edges and graph topology, containing at least the columns "ID", "N1", "N2" and "Value", specifying, respectively, the ID of the edge, the parent node, the child node, and the size of the edge.

styles A named list of styles. Names of this list are the node or edge IDs. Values are styles specifying the style of the given node or edge (see below).

Whether or not the list used to plot is exactly of class riverplot-class does not matter as long as it has the correct contents. The makeRiver function is here are for the convenience of checking that this is the case and converting information in different formats.

Generating riverplot objects

To generate and fool-proof riverplot objects, you can use the makeRiver function. This functions allows a number of ways of specifying the node and edge information.

Nodes can be specified as a character vector (simply listing the nodes) or as a data frame.

- character vector: in this case, you also need to provide the *node_xpos* argument to specify the horizontal positions of the nodes.
- data frame: the data frame must have at least a column called "ID"; the horizontal position can be specified either with *node_xpos* argument or by column "x" in the data frame. Optionally, the data frame can include columns "labels" and "y" (vertical positions of the node). Any *NA* values are ignored (not entered into the riverplot project). Additionaly, the data frame may contain style information.

Edges / graph topology can be specified in one of two objects: either a named list, or a data frame:

- you can supply a named list with edges of the graph. The name of each element is the name of the outgoing (parental) node. Each element is a named list; the names of the list are the names of the incoming (child) node IDs; the values are the width of the edge between the outgoing and incoming nodes.
- Alternatively, you can provide the edges as a data frame. Each row corresponds to an edge, and the data frame must have the following columns:

  N1 The ID of the first node

  N2 The ID of the second node

  Value The width of the edge between N1 and N2

  If an ID column is absent, it will be generated from N1 and N2 by joining the N1 and N2 ID's with the "->" string. Additionaly, the data frame may contain style information. Any *NA* values are ignored (not entered into the riverplot object).

Riverplot styles

Styles are lists containing attributes (such as "col" for color or "nodestyle") and values. There is no real difference between node and edge styles, except that some attributes only apply to nodes or edges. See `riverplot-styles` for more information on style attributes.

When `makeRiver` generates the riverplot object, it combines style information from the following sources in the following order:

- parameter *default_style* is a style applied to all nodes and edges
- if the parameter *nodes* and/or *edges* is a data frame, it may include columns with names corresponding to style attributes. For example, a column called "col" will contain the color attribute for any nodes / edges. *NA* values in these columns are ignored.
- *styles* is a lists of styles, with names corresponding to node IDs or edge IDs, which will replace any previously specified styles.

Author(s)

January Weiner

Examples

```
nodes <- c( LETTERS[1:3] )
edges <- list( A= list( C= 10 ), B= list( C= 10 ) )
r <- makeRiver( nodes, edges, node_xpos= c( 1,1,2 ),
  node_labels= c( A= "Node A", B= "Node B", C= "Node C" ),
  node_styles= list( A= list( col= "yellow" )) )
plot( r )

# equivalent form:
nodes <- data.frame( ID= LETTERS[1:3],
                x= c( 1, 1, 2 ),
                col= c( "yellow", NA, NA ),
                labels= c( "Node A", "Node B", "Node C" ),
                stringsAsFactors= FALSE )
r <- makeRiver( nodes, edges )
```

```
plot( r )
# all nodes but "A" will be red:
r <- makeRiver( nodes, edges, default_style= list( col="red" ) )
plot( r )
# overwrite the node information from "nodes":
r <- makeRiver( nodes, edges, node_styles= list( A=list( col="red" ) ) )
plot( r )
```

`minard` *Minard Napoleon Russian campaign data*

Description

The data set used by Charles Joseph Minard to generate the famous graph. The example below shows how to recreate the main panel of the graph using riverplot from the provided data.

Usage

```
minard
```

Format

Named list with two data frames:

nodes data frame with geographic locations of the Napoleon army (longitude and latitude) and the direction of the march

edges connections between positions

Details

First, node and edge data frames must get new column names (see `makeRiver` function for details). Then, based on the direction of the Napoleon army, style information (right and left edge color style for each node) is entered in the *nodes* variable. Then, a riverplot object is generated from the nodes and edges data frames.

To use the same color coding as Minard, the *direction* variable is converted to color codes in the *col* column of the *edges* object.

Finally, a plot is created using `lty=1` and a style in which nodes are not shown, and the edges are straight (like in the original Minard plot) rather than curved.

Author(s)

January Weiner

Source

Charles Joseph Minard

Examples

```
# example how to convert data into a riverplot object
data(minard)
nodes <- minard$nodes
edges <- minard$edges
colnames(nodes) <- c("ID", "x", "y")
colnames(edges) <- c("N1", "N2", "Value", "direction")

# color the edges by troop movement direction
edges$col <- c("#e5cbaa", "black")[factor(edges$direction)]

# color edges by their color rather than by gradient between the nodes
# The "edgecol" column is interpreted as a style keyword with value "col"
edges$edgecol <- "col"

# generate the riverplot object and a style
river <- makeRiver(nodes, edges)
style <- list(edgestyle= "straight", nodestyle= "invisible")

# plot the generated object. Given that we want to plot the cities as well
# (external data), the user coordinates for the plot and for the external
# data should be the same. This is achieved by the adjust.usr option.
# Alternatively, one can call plot.new, set usr manually and call riverplot
# with the options rescale=FALSE and add=TRUE.
# plot_area parameter is for creating suitable margins within the plot area
par(bg="grey98", mar=rep(3,4))
plot(river, lty=1, default_style=style, plot_area=c(0.9, 0.7), adjust.usr=TRUE)
u <- par("usr")
rect(u[1], u[3], u[2], u[4])

# add latitude and longitude
abline(h=54:56, col="grey")
bglabel(u[1], 54:56, sprintf("%d°N", 54:56), pos="topright", bg=NA, col="grey", font=3)
lbl <- seq(20, 40, by=5)
abline(v=lbl, col="grey")
bglabel(lbl, u[3], sprintf("%d°E", lbl), pos="topright", bg=NA, col="grey", font=3)

# Add cities. Use "bglabel()" to have a background frame and better
# positioning.
with(minard$cities, points(Longitude, Latitude, pch=19))
with(minard$cities, bglabel(Longitude, Latitude, Name, pos="topright"))
```

`plot.riverplot` *Create a Sankey plot*

Description

Create a Sankey plot

Usage

```
## S3 method for class 'riverplot'
plot(x, ...)

riverplot(
  x,
  direction = "lr",
  lty = 0,
  default_style = NULL,
  gravity = "top",
  node_margin = 0.1,
  nodewidth = 1.5,
  plot_area = c(1, 0.5),
  nsteps = 50,
  disentangle = TRUE,
  add_mid_points = TRUE,
  yscale = "auto",
  add = FALSE,
  usr = NULL,
  adjust.usr = FALSE,
  rescale = TRUE,
  fix.pdf = FALSE,
  bty = "n",
  ...
)
```

Arguments

x	An object of class riverplot
...	any further parameters passed to riverplot() are appended to the default style
direction	"lr" (left to right) or "rl" (right to left)
lty	Line style to use
default_style	default graphical style
gravity	how the nodes are placed vertically. No effect if node vertical positions are specified via *node_ypos* member
node_margin	how much vertical space should be kept between the nodes
nodewidth	width of the node (relative to font size)
plot_area	fraction of vertical and horizontal space to be used as main plot area If it is a numeric vector of two numbers, the first one is horizontal space, the second vertical.
nsteps	number of interpolating steps in drawing the segments
disentangle	try to disentangle connections between the nodes. If FALSE, the vertical ordering of the connections is the same as in the x$edges data frame.
add_mid_points	attempt to get a smoother plot by adding additional nodes. Set this parameter to FALSE if you are setting node vertical position manually. If add_mid_points is

equal to TRUE (the default), then the mid points are added only if *node_ypos* is empty.

`yscale` scale the edge width values by multiplying with this factor. If *yscale* is equal to "auto", scaling is done automatically such that the vertical size of the largest node is approximately 15 If no *node_ypos* is specified in the riverplot object, no scaling is done. If *yscale* is equal to 1, no scaling is done. This parameter only influences the plot if the y positions of the nodes are provided in `x$nodes`.

`add` If TRUE, do not call plot.new(), but add to the existing plot.

`usr` coordinates at which to draw the plot in form (x0, x1, y0, y1). If NULL, par("usr") will be used instead.

`adjust.usr` If TRUE, the par("usr") will be modified to suit the x and y coordinates of the riverplot nodes (whether the coordinates were given in the nodes, or calculated by the function). In combination with providing x and y coordinates, this allows a true representation of a riverplot object. Necessary if you plan to plot additional, external data. If TRUE, then `rescale` is set to FALSE. See `minard` data set and example for details.

`rescale` if TRUE, then the plot will be fit into the given user coordinates range (set by the usr parameter, for example, or the whole plot region). If FALSE, the x and y positions of the nodes will be treated as user coordinates and used to directly plot on the device.

`fix.pdf` Try to fix PDF output if it looks broken (with thin white lines). Don't use this option if you are using transparent colors.

`bty` box type to draw around the plot; see `bty` in documentation for `par` for details.

Details

This functions create a Sankey plot given a riverplot object (`plot` is just a wrapper for the `riverplot` function. The object to be drawn is a list specifying the plot; see the `makeRiver` function for exact specifications and the `riverplot.example` to see how it can be created. Whether or not the list used to plot is exactly of class `riverplot-class` does not matter as long as it has the correct contents.

Style information which is missing from the riverplot object `x` (for example, if the node style is not specified for each node in the object) is taken from the `default.style` parameter. See functions `default.style()` and `updateRiverplotStyle()` to learn how to create and modify the styles.

Whether or not the list used to plot is exactly of class `riverplot-class` does not matter as long as it has the correct contents. These functions here are for the convenience of checking that

The nodes are drawn from bottom to top in the order they are found in the riverplot object. There is no clever algorithm for placing the nodes minimizing the number of crossing edges yet; you need to manipulate the object directly to achieve the desired effect.

Value

`riverplot` returns a riverplot object, a graph which you can plot again with riverplot(), but which additionally contains information on node position and size in the `$nodes` member.

Known problems

There is a problem with transparency and PDFs. In short, if you try to save your riverplot graphics as PDF, you will observe thin, white vertical lines everywhere on the curves. The reasons for that are unclear, but have something to do with PDF rendering (if you generate EPS, the output looks good).

There is a kind of fix to that: use the fix.pdf=TRUE option. Unfortunately, this solution does not work if you use transparent colors (you will have a different kind of vertical lines). Unfortunately, I don't have a solution for that problem yet.

See Also

default.style updateRiverplotStyle minard

Examples

```
x <- riverplot.example()
plot(x)
plot(x, srt=90, lty=1)

# add graphics at nodes
foo <- plot(x, srt=90, lty=1)
points(foo$nodes$x, foo$nodes$y, pch=19, cex=2)

# redraw the same graph using positions from foo object
plot(foo, yscale=1)
```

---

riverplot-styles *Riverplot styles*

Description

Riverplot styles

Usage

```
default.style()

updateRiverplotStyle(style, master)
```

Arguments

style	style to update
master	master style to use for updating

Details

Riverplot styles are just lists with key-value pairs that define how nodes and edges are drawn. Although there are attributes that are only applicable to either nodes or edges, there are no separate style lists for these objects.

The `default.style` function simply returns the default style defined in the riverplot package (including edge and node attributes).

The `updateRiverplotStyle` function updates all missing fields in the `style` object with the styles from the `master` style.

When a node is drawn, the styles are determined by precedence. Command line arguments to `riverplot()` function override any defined styles. For all other parameters styles associated with nodes are used, and if absent, inserted from the `default.style` argument to the `riverplot()` function. If this argument is missing, style is taken from the argument returned by the `default.style` function.

Not recognized fields and values will be silently ignored.

Following style fields and values are defined:

nodestyle (default: regular). Values:

- **regular** rectangular box with a label
- **point** a color dot
- **invisible** No node is drawn. This is used to seamlessly integrate edges.

edgestyle (default: sin). Describes how the edge looks like.

- **sin** A sinusoidal edge
- **straight** A straight edge

edgecol (default: "gradient"). How edge color is generated. Values:

- **gradient** A color gradient generated based on parent and child node that form the edge
- **col** The color specified in the "col" attribute of the edge

horizontal (default: `FALSE`). If set to `TRUE`, the edge will be drawn horizontally by repositioning the node on the right hand side. This may mess up the figure, so beware.

col (default: "grey"). Color of the node or edge (for edges, it is used only if the "edgecol" attribute is "col".

srt (default: "90"). Rotation of the label (see `par`)

lty (default: 1). Line type to draw around node and edges

textcol (default: "black"). Color of the node label.

textpos (default: NULL). Label position, passed on to "pos" argument of the text() function.

textcex (default: 1). Label cex, passed on to "cex" argument of the text() function.

Value

Both functions return an object of the riverplotStyle class (which is, in fact, just a list with key-value pairs that you can access, inspect and manipulate manually at will).

Author(s)

January Weiner

Examples

```
# To view the default style specification, type
default.style()

ex <- riverplot.example()
ds <- default.style()
plot( ex, default_style= ds )

# nodes with unspecified style will now be semi-transparent red:
ds[["col"]] <- "#FF000099"
plot( ex, default_style= ds )
```

`riverplot.example` *Generate examples for riverplot*

Description

Generate an example for riverplot

Usage

```
riverplot.example(no = 1)
```

Arguments

`no`	which example to generate

Details

The plotting functions in the riverplot package work on an object of the riverplot class. This function returns an object of the riverplot class to demonstrate how such an object (which is actually a simple list) can be created.

Author(s)

January Weiner <january.weiner@gmail.com>

Examples

```
x <- riverplot.example()
plot( x )
x <- riverplot.example(no=2)
riverplot(x, lty=1, plot_area=1, disentangle=TRUE,
   gravity="c", default_style=list(nodestyle="invisible"))
```

Index