riverplot: A Gallery

January Weiner

2017-04-24

Abstract

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.

Contents

Introduction

This document consists of two parts. The first part shows a few more or less complex examples with a few explanations of the code. The second, which is visually grey and boring, elaborates on using the different graphical parameters of riverplot. Bottom line: I put the examples first, because they are more interesting, but if you really want to fiddle with the finer points of the graphics, you need to refer to part 2.

Part 1: Gallery

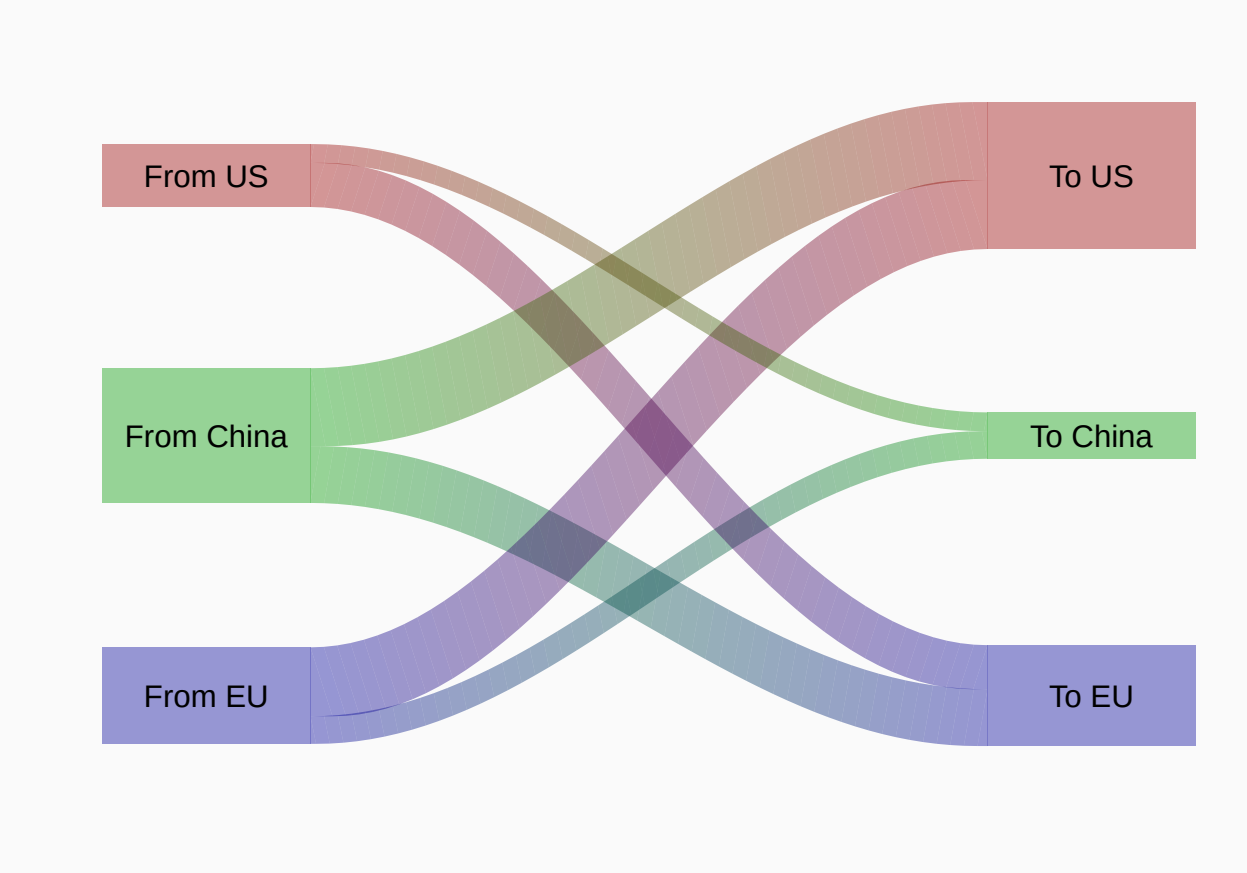
Import and export of goods

For this example, I have selected a few

```r
options(stringsAsFactors=FALSE)
## goods exports / imports for 2015 in billions
edges <- read.csv(text="N1,N2,Value
From EU,To China,170.4
From China,To EU,350.6
From EU,To US,426.0
From US,To EU,272.7
From China,To US,482
From US,To China,116",
header=T, stringsAsFactors=FALSE)
print(edges)
```

```
##           N1       N2 Value
## 1    From EU To China 170.4
## 2 From China    To EU 350.6
## 3    From EU    To US 426.0
## 4    From US    To EU 272.7
## 5 From China    To US 482.0
## 6    From US To China 116.0
```

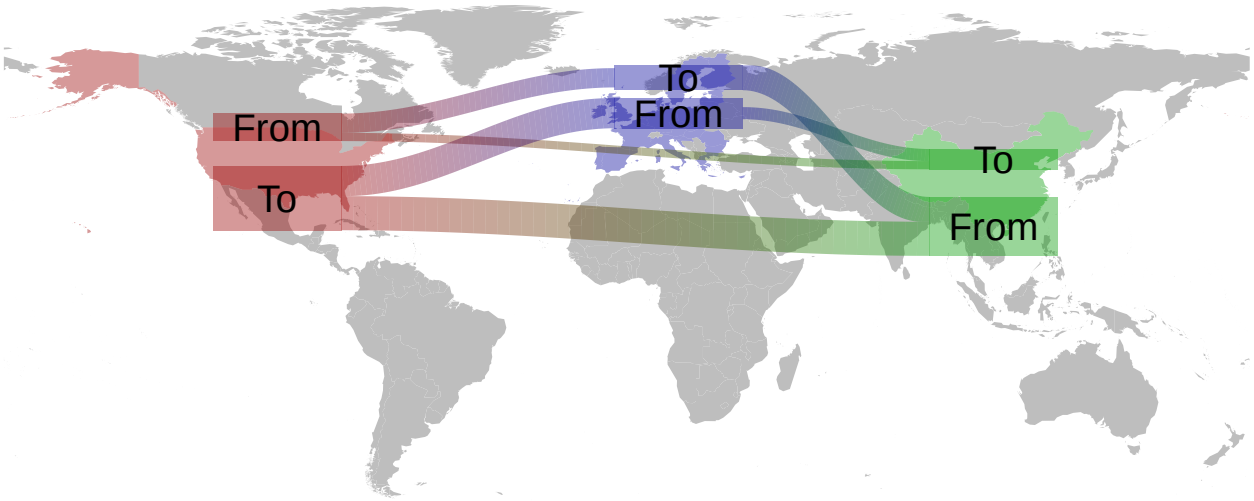
```r
nodes <- data.frame(ID=unique(c(edges$N1, edges$N2)),
    x=rep(c(1,2), each=3), y=c(1,2,3,2,1,3))

cols <- c(China="#00990066",
          EU   ="#00009966",
          US   ="#99000066")

style <- sapply(nodes$ID, function(id)
  list(col=cols[ gsub("(From|To) ", "", id) ]), simplify=FALSE)

r <- makeRiver(nodes=nodes, edges=edges, styles=style)
par(bg="grey98")
d <- list(srt=0, textcex=1.5) # default style
plot(r, plot_area=1, nodewidth=10, default_style=d)
```



In the second example, the goal is to put the same information on a geographical map. For this, first a map should be shown and colored, and then the Sankey diagram should be overlayed at prespecified positions.

```
eu <- c( "Austria", "Italy", "Belgium", "Latvia", "Bulgaria",
"Lithuania", "Croatia", "Luxembourg", "Cyprus", "Malta", "Czech Republic",
"Netherlands", "Denmark", "Poland", "Estonia", "Portugal", "Finland", "Romania",
"France", "Slovakia", "Germany", "Slovenia", "Greece", "Spain", "Hungary", "Sweden",
"Ireland", "United Kingdom")

library(maptools)
data(wrld_simpl)
m <- wrld_simpl[ wrld_simpl@data$NAME != "Antarctica", ]

nodes$labels <- gsub(" .*", "", nodes$ID)
nodes$x <- c( 15, 106, -101,
              106,  15, -101)

nodes$y <- c(53, 21, 49,
             40, 63, 29 )
ccol <- rep("grey", nrow(m@data))
ccol[ m@data$NAME == "China" ] <- cols["China"]
ccol[ m@data$ISO2 == "US" ] <- cols["US"]
ccol[ m@data$NAME %in% eu ] <- cols["EU"]

r <- makeRiver(nodes=nodes, edges=edges, styles=style)
plot(m, border=F, col=ccol)
```

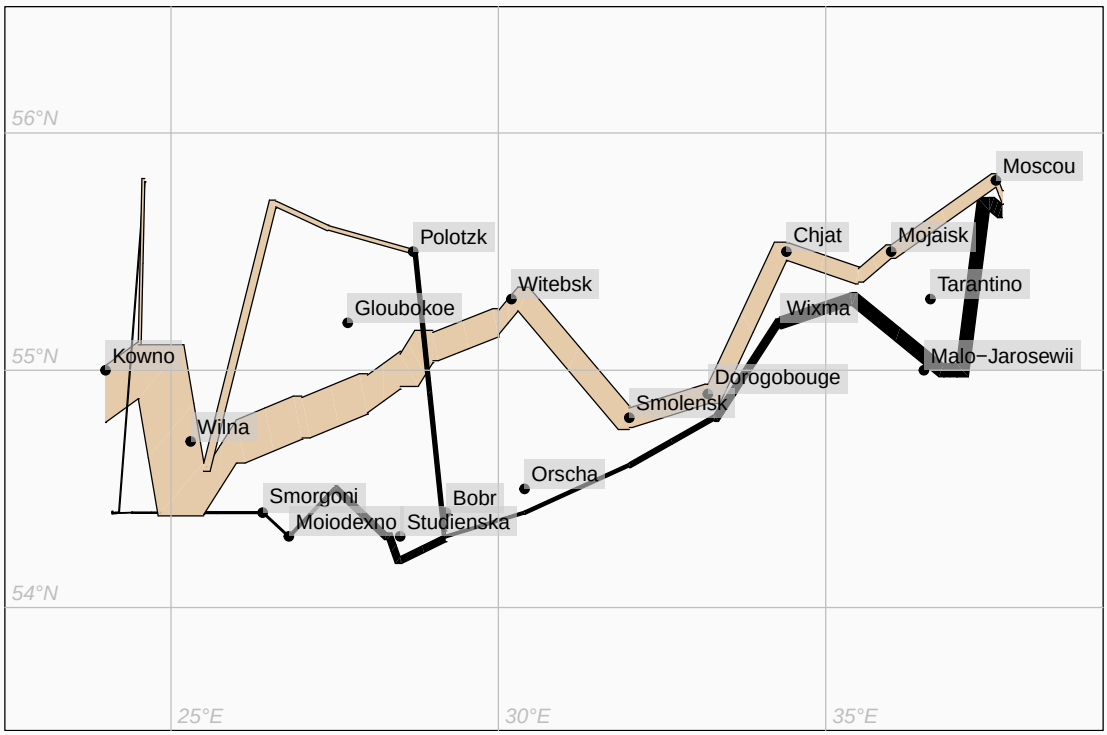
```
d <- list(srt=0, textcex=1.5)
plot(r, add=T, rescale=F, yscale=0.02, default_style=d, nodewidth=5)
```



The famous Minard plot

```
library(riverplot)
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 "label()" 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"))
```



Part 2: Customizing plots with riverplot

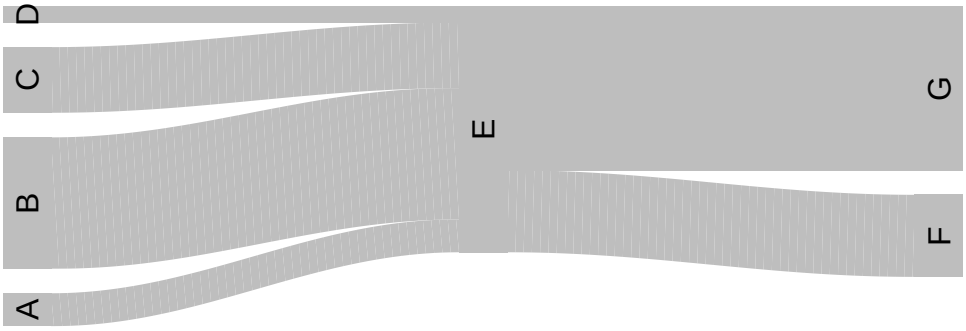
A first example

Sankey plots are used to illustrate the flow of quantities. In riverplot, sankey plots consist of nodes and edges. Edges always have a defined breadth – they do not dwindle or broaden when going from node to node.

In order to use riverplot, an object holding the data must first be defined. The object can be generated manually or with `makeRiver`, a function that additionally checks whether the information in the object is consistent and correct.

We start with a simple object with a few nodes and edges. There are many ways how the object can be constructed (consult the manual for `makeRiver` for that), but data frames for nodes and edges are probably the easiest:

```
library(riverplot)
options(stringsAsFactors=FALSE)
nodes <- data.frame(ID=LETTERS[1:7], x=c(1, 1, 1, 1, 2, 3, 3))
edges <- data.frame(ID=paste0("E.", letters[1:6]),
        N1=c("A", "B", "C", "D", "F", "G"),
        N2=c("E", "E", "E", "E", "E", "E"),
        Value=c(10, 40, 20, 5, 25, 50))
par(mar=rep(0,4))
r <- makeRiver(nodes, edges)
plot(r)
```

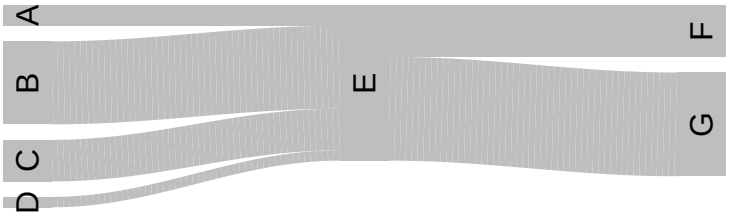


In the following, it will be shown how to customize this simple plot.

Fixing the order of nodes

The order of nodes matters. As you can see above, the nodes are stacked from bottom to top using the order in which they were defined in the `nodes` data frame. Reversing this order will result in a different picture:

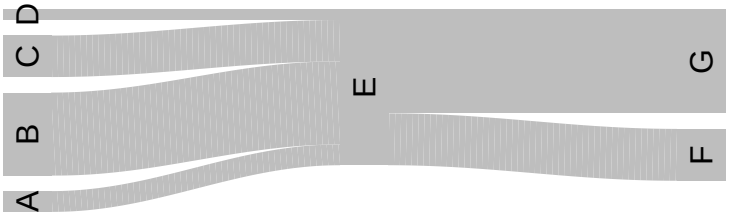
```
nodes2 <- nodes[7:1,]
r2 <- makeRiver(nodes2, edges)
plot(r2)
```



Pre-specified x and y values

A finer control can be achieved by directly providing (pinning) positions of the nodes on the figure. The easiest way to achieve this is by letting riverplot generate the positions in the first place, and then tuning them according to own preferences:

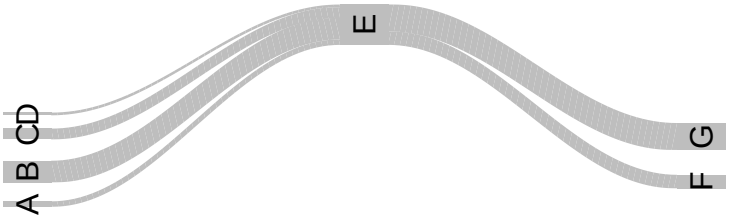
```
r2 <- plot(r)
```




```
str(r2$nodes) # nodes contain now column "y"
```

```
## 'data.frame':    7 obs. of  6 variables:
##  $ ID   : chr  "A" "B" "C" "D" ...
##  $ x    : num  -0.00408 -0.00408 -0.00408 -0.00408 0.5 ...
##  $ size : num  0.0369 0.1477 0.0738 0.0185 0.2769 ...
##  $ sizeL: num  0 0 0 0 0.277 ...
##  $ sizeR: num  0.0369 0.1477 0.0738 0.0185 0.2769 ...
##  $ y    : num  0.338 0.458 0.597 0.671 0.542 ...
```

```
r2$nodes["E", "y"] <- 1
plot(r2)
```

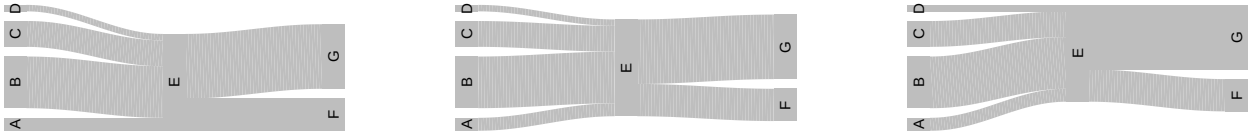


The node "E" is now elevated well above the other nodes. However, note that the relative node widths have changed; this is because of the `yscale` parameter – see below.

Vertical alignment using paramter `gravity`

The way nodes and edges are stacked on the plots depends on the parameter `gravity`. Depending on it, the nodes will be sticking to the bottom, top or center of the figure:

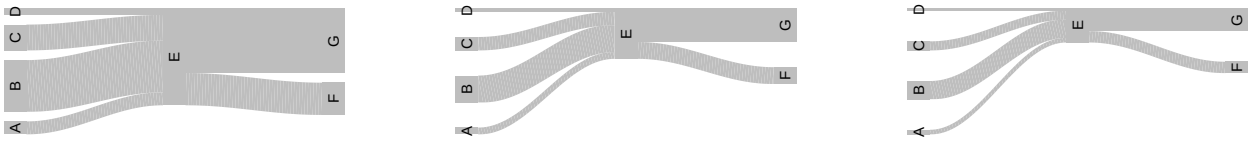
```
par(mfrow=c(1,3))
plot(r, gravity="b")
plot(r, gravity="c")
plot(r, gravity="t") # default
```



Controlling the node width on the picture (parameter `yscale` and `node_margin`)

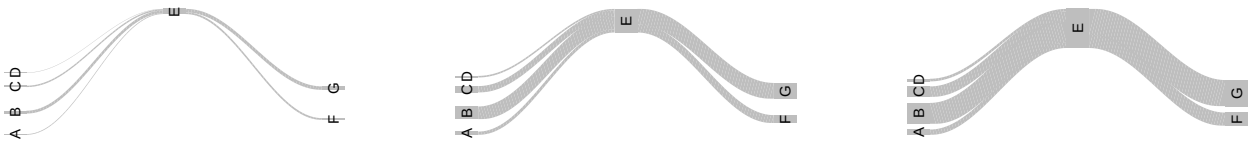
Essentially, riverplot runs in one of two modes. By default, you don't need to provide an exact location of all nodes generated by riverplot. In this case, the width of the edges shown on the figure results from two factors: the amount of space available for plotting and the margin between the nodes (parameter `node_margin`), as the fraction of all available space.

```
par(mfrow=c(1,3))
plot(r, node_margin=0.1) # default
plot(r, node_margin=0.5)
plot(r, node_margin=0.9)
```



However, if the node y positions are specified (by the column `y` of the `nodes` data frame), another paramter controls the width of the nodes: `yscale` (normally set to "auto").

```
par(mfrow=c(1,3))
plot(r2, yscale=0.1)
plot(r2, yscale=0.5)
plot(r2, yscale=0.9)
```

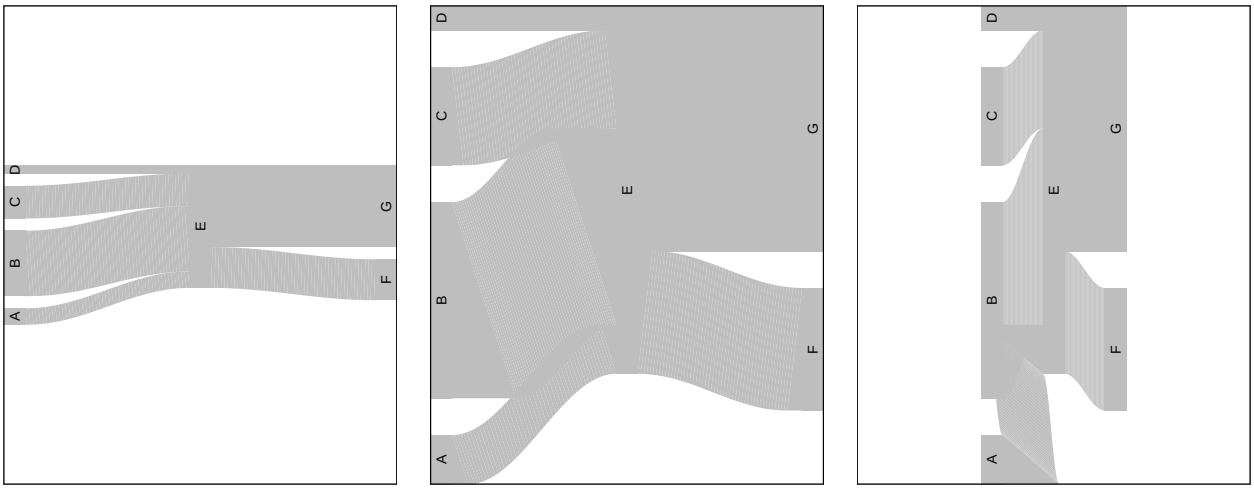


Specifying the plotting area with `plot_area`

Sometimes it is useful to impose a certain geometry of the plot. This can be, of course, achieved with changing the margins (like with `par(mar=...)` above), the size of the plotting device etc. However, additionally the parameter `plot_area` or `usr` (see below) can be used. The parameter `plot_area` specifies both the horizontal and the vertical fraction of the space used by the plot, either as two separate numbers or as one number for both these fractions. By default, riverplot uses 100% of the horizontal space and 50% of the vertical space, so `plot_area` is `c(1, 0.5)`.

We use additionally the parameter `bty="o"` to draw a box around the plot.

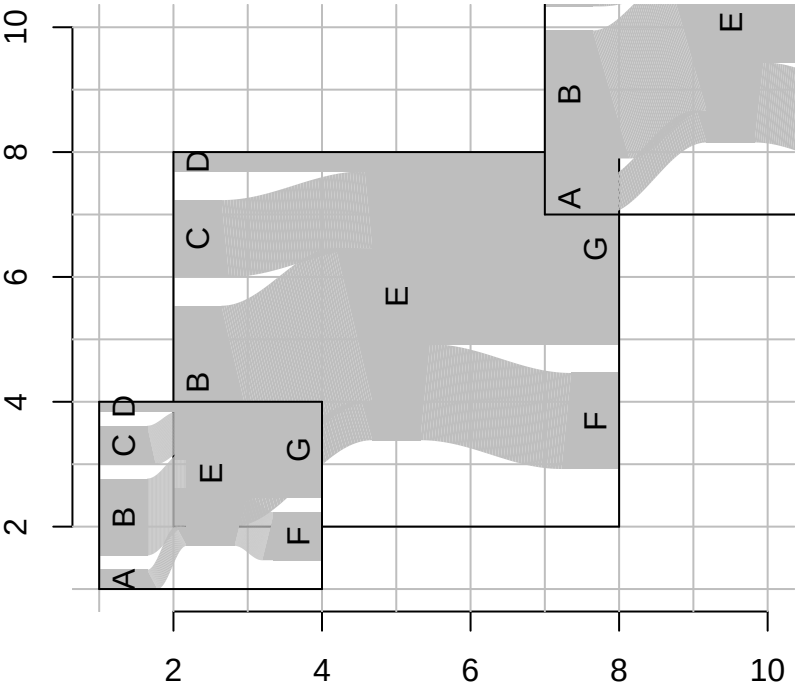
```
par(mfrow=c(1,3))
par(mar=rep(1,4))
plot(r, plot_area=c(1, 0.5), bty="o") # default
plot(r, plot_area=1, bty="o") # max area in both directions
plot(r, plot_area=c(0.5, 1), bty="o") # squeezed laterally
```



Specifying the plotting area with `usr` and overlaying riverplots

A more finely tuned control of the area of the plot is achieved with the parameter `usr`, which has the same syntax as `par(usr=...)`; that is, it give the coordinates of the bounding box of the plot in the form of `c(x1, x2, y1, y2)`. Combined with the parameter `add=TRUE`, this allows to place a sankey diagram on top of another diagram in a specified box.

```
plot(NULL, xlim=c(1,10), ylim=c(1,10), bty="n", xlab="", ylab="")
abline(h=1:10, col="grey")
abline(v=1:10, col="grey")
plot(r, add=TRUE, usr=c(2,8,2,8), plot_area=1, bty="o")
plot(r, add=TRUE, usr=c(1,4,1,4), plot_area=1, bty="o")
plot(r, add=TRUE, usr=c(7,12,7,12), plot_area=1, bty="o")
```



Note that while `usr` sets the boundary box of the Sankey diagram, the `plot_area` still controls the area of the box that is occupied by the diagram. In the code above, the `plot_area` is set to 1 to occupy the whole bounding box.

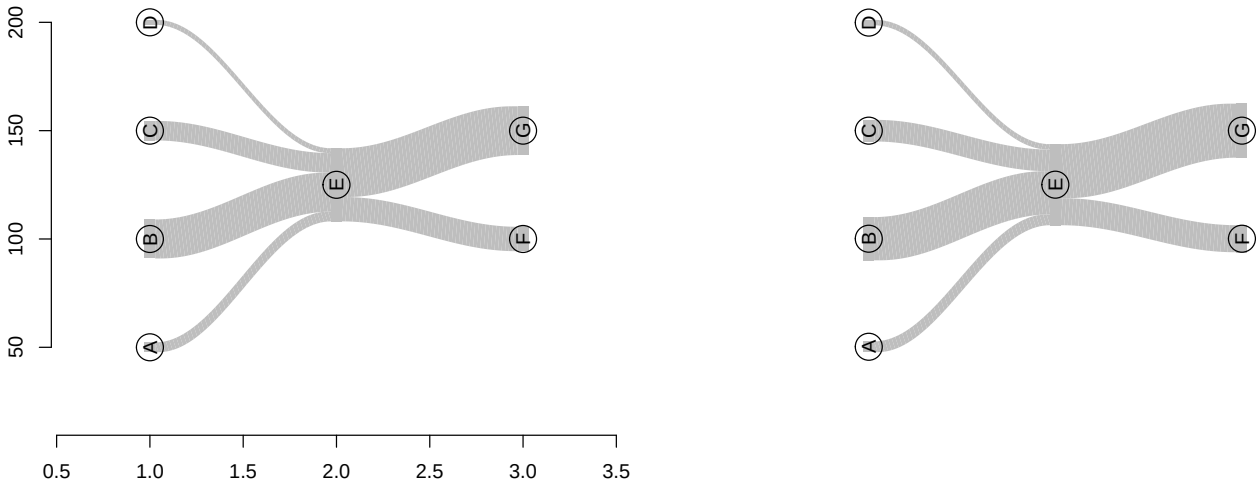
More about adjusting the `usr` parameter (`adjust.usr=TRUE`) and `yscale`

In general, riverplot does not modify the `par("usr")` parameter, and instead adjusts the x and y coordinates of the nodes as well as widths (`Value`) of the edges. This actually is the opposite of the typical plotting behavior in R, but makes it easier to add the Sankey diagram to existing plots.

However, sometimes it is desired to plot precisely using the coordinates specified by the graph, for example, because one wishes to add additional, external information to the figure. This is, for example, the case of the Minard data set, in which the node coordinates should correspond to external data – geographical coordinates of the cities added to the plot after plotting the Sankey diagram (see the Minard example in the gallery).

Specifying `adjust.usr=TRUE` modifies the `usr` parameter by setting `par("usr")`.

```
par(mfrow=c(1,2))
nodes2 <- nodes[order(nodes$ID),]
nodes2$y <- c(50*(1:4),125,100,150)
r2 <- makeRiver(nodes2, edges)
plot(r2, adjust.usr=TRUE, plot_area=c(0.8,0.8))
points(nodes2$x, nodes2$y, cex=3)
axis(side=1)
axis(side=2)
plot(r2, adjust.usr=TRUE, plot_area=c(0.8,0.8), yscale=0.5)
points(nodes2$x, nodes2$y, cex=3)
```



Note that if `adjust.usr` is `TRUE`, and `yscale` is `auto` (default), then `yscale` is set to 1. That means, the `Value` of edges is directly representing their thickness. If the `Value` of an edge was 20, then it will be 20 thick (in user coordinates). However, you can still manipulate it by setting another `yscale`. This is shown on the right panel above.

Note also that `adjust.usr` also implies that `rescale` is `FALSE` (it would make little sense otherwise).

Problematic output with PDFs (`fix.pdf` option)

Sometimes you will see thin white lines on the PDF versions of your plots. The reason for this is in rendering of the PDFs, not in the code of riverplot or R itself, but it is annyoing.

R cannot generate a "true" gradient (whatever that would be). To create a gradient, riverplot creates objects which consist of hundreds of smaller objects, each of them being a polygon with a single foreground color. This usually looks good on the standard R device; usually the produced bitmap graphics (such as PNG) looks fine as well. However, the created PDFs may look very awkward.

If you use the option `fix.pdf=TRUE` when generating riverplots, the individual polygons will be slightly overlapping. This will fix the "thin white lines" effect, but only providing that you do not use transparent colors.