Package 'genoPlotR'

September 17, 2010

Type Package

Title Plot publication-grade gene and genome maps

Version 0.5.3

Date 2009-12-08

Author Lionel Guy <lionel.guy@ebc.uu.se>

URL http://genoplotr.r-forge.r-project.org/

Depends R (>= 2.10.0), methods, grid, ade4

Maintainer Lionel Guy <lionel.guy@ebc.uu.se>

Description genoPlotR draws gene or genome maps and comparisons between these, in a publication-grade manner. Starting from simple, common files, it will draw postscript or pdf files that can be sent as such to journals

License GPL (>= 2)

LazyLoad yes

R topics documented:

`genoPlotR-package` *genoPlotR - a R framework to produce publication-grade maps of genes and genomes.*

Description

A R framework to plot comparison of gene stretches or genomes, a la ACT (Artemis Comparison Tool), but with production-grade graphics, and a static interface. Reads directly from tabular files or from wide-spread biological formats such as BLAST and PTT (NCBI).

Details

Package:	genoPlotR
Type:	Package
Version:	0.1
Date:	2009-12-08
License:	GPL (>=2)
LazyLoad:	yes

The only plotting function is `plot_gene_map`, which produces `link[grid]{grid}` graphics. Data is composed mainly of DNA segments (`dna_seg`) objects, which represent collections of genes or segments of genomes, and of `comparison` objects, which are the pairwise comparisons between the `dna_segs`. Data can be read from files (see `read_functions`) or from R objects like `data.frames` or `lists`, with `dna_seg` and `comparison` conversion functions.

Author(s)

Lionel Guy

Maintainer: Lionel Guy `<lionel.guy@ebc.uu.se>`

See Also

`plot_gene_map` for plotting. `dna_seg` and `comparison` for the base objects and conversion functions. `read_dna_seg_from_tab`, `read_dna_seg_from_ptt`, `read_comparison_from_tab` and `read_comparison_from_blast` to read from files.

Examples

```
## simple example
## dna segments
## data.frame with several genes
names1 <- c("feat1", "feat2", "feat3")
starts1 <- c(2, 1000, 1050)
ends1 <- c(600, 800, 1345)
strands1 <- c("-", -1, 1)
cols1 <- c("blue", "grey", "red")
df1 <- data.frame(name=names1, start=starts1, end=ends1,
                  strand=strands1, col=cols1)
dna_seg1 <- dna_seg(df1)
```

```
is.dna_seg(dna_seg1)

## with only one gene, or two, and merging
gene2a <- dna_seg(list(name="feat1", start=50, end=900, strand="-", col="blue"))
genes2b <- dna_seg(data.frame(name=c("feat2", "feat3"), start=c(800, 1200),
                              end=c(1100, 1322), strand=c("+", 1),
                              col=c("grey", "red")))
dna_seg2 <- c.dna_seg(gene2a, genes2b)
is.dna_seg(dna_seg2)

## reading from file
dna_seg3_file <- system.file('extdata/dna_seg3.tab', package = 'genoPlotR')
dna_seg3 <- read_dna_seg_from_tab(dna_seg3_file)
is.dna_seg(dna_seg3)

## comparison
## from a data.frame
comparison1 <- as.comparison(data.frame(start1=starts1, end1=ends1,
                                        start2=dna_seg2$start,
                                        end2=dna_seg2$end))
is.comparison(comparison1)

## from a file
comparison2_file <- system.file('extdata/comparison2.tab',
                                package = 'genoPlotR')
comparison2 <- read_comparison_from_tab(comparison2_file,
                                        color_scheme="red_blue")
is.comparison(comparison1)

## plot
plot_gene_map(dna_segs=list(dna_seg1, dna_seg2, dna_seg3),
              comparisons=list(comparison1, comparison2))
```

`annotation` *Annotation class and class functions*

Description

An annotation describes a DNA segment. It has labels attached to positions. Each label can be attached to a single position or to a range.

Usage

```
annotation(x1, x2 = NA, text, rot = 0, col = "black")
as.annotation(df, x2 = NA, rot = 0, col = "black")
is.annotation(annotation)
```

Arguments

`x1`	Numeric. A vector giving the first or only position of the label. Mandatory.
`x2`	Numeric. A vector of the same length as `x1`. If a row (or the whole column is `NA`, then the annotation(s) will be attached to `x0`. Else, the annotation will be attached to the range between both positions. `NA` by default.

text	Character of the same length as x0. Gives the text of the labels. Mandatory.
rot	Numeric of the same length as x0. Gives the rotation, in degrees, of the labels. 0 by default.
col	Vector of the same length as x0. The color of the labels. black by default.
df	A data frame to convert to an annotation object. Should have at least columns x1 and text.
annotation	An object to test.

Details

An annotation object is a data frame with columns x0, x1, text, col and rot. They give, respectively, the first (or only) position, eventually the second position, the text, the color and the rotation of the annotation. When plotted with plot_gene_map, it will add an annotation row on top of the first dna_seg. Labels for which only one position is given will be centered on that position. Labels for which two positions are given are linked by an horizontal square bracket and the label is plotted in the middle of the positions.

Value

annotation and as.annotation return an annotation object. is.annotation returns a logical.

Author(s)

Lionel Guy

See Also

plot_gene_map, middle.

Examples

```
## loading data
data(three_genes)

## Calculating middle positions
mid_pos <- middle(dna_segs[[1]])

# Create first annotation
annot1 <- annotation(x1=mid_pos, text=dna_segs[[1]]$name)
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons, annotations=annot1)

## Exploring options
annot2 <- annotation(x1=c(mid_pos[1], dna_segs[[1]]$end[2]),
                     x2=c(NA, dna_segs[[1]]$end[3]),
                     text=c(dna_segs[[1]]$name[1], "region1"),
                     rot=c(30, 0), col=c("grey", "black"))
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              annotations=annot2, annotation_height=1.3)

## Annotations on all the segments
annots <- lapply(dna_segs, function(x){
  mid <- middle(x)
  annot <- annotation(x1=mid, text=x$name, rot=30)
```

```
})
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              annotations=annots, annotation_height=1.8, annotation_cex=1)
```

`apply_color_scheme` *Apply a color scheme*

Description

Apply a color scheme to a numeric vector, eventually taking the direction into account.

Usage

```
apply_color_scheme(x, direction = NULL, color_scheme = "grey",
decreasing = FALSE, rng = NULL)
```

Arguments

`x`	A numeric, that will be used to apply a gradient of colors to a comparison.
`direction`	If a red-blue scheme is choosen, the vector (composed of -1 and 1 values and of same length as `x`) giving the direction of the comparison.
`color_scheme`	Character. One of `red_blue`, `blue_red`, `grey`, `gray`.
`decreasing`	Logical. Are the values of the comparisons oriented such as the lower the value, the closer the relationship (e.g. e-values, gaps, mismatches, etc)? `FALSE` by default.
`rng`	Numeric of length 2. Gives the higher and lower limit to apply a color scheme.

Details

A color scale is calculated, with the darker color corresponding to the highest values of `x`, or the contrary is `decreasing` is `TRUE`. For the moment, two schemes (red-blue and grey scale) are used.

For the red-blue scale (as in ACT), the direct comparisons are colored in red hues, and the reversed ones in blue hues.

This is especially useful to replace comparison values (such as BLAST percent identity values) by color hues.

Value

A character vector of same length as `x`, representing colors.

Author(s)

Lionel Guy

References

Artemis Comparison Tool, http://www.sanger.ac.uk/Software/ACT/

See Also

comparison

Examples

```
## Load data
data(three_genes)

## Color schemes
## Greys
comparisons[[1]]$values <- c(70, 80, 90)
comparisons[[1]]$col <- apply_color_scheme(comparisons[[1]]$values,
                                           color_scheme="grey")
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons)
## Red-blue
comparisons[[1]]$col <- apply_color_scheme(comparisons[[1]]$values,
                                           direction=comparisons[[1]]$direction,
                                           color_scheme="red_blue")
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons)
## Decreasing
comparisons[[1]]$col <- apply_color_scheme(comparisons[[1]]$values,
                                           direction=comparisons[[1]]$direction,
                                           color_scheme="red_blue",
                                           decreasing=TRUE)
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons)
## Range
comparisons[[1]]$col <- apply_color_scheme(comparisons[[1]]$values,
                                           direction=comparisons[[1]]$direction,
                                           color_scheme="red_blue",
                                           rng=c(30,100))
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons)
```

---

barto *Comparison of 4 Bartonella genomes*

Description

Comparison of 4 Bartonella genomes by BLAST.

Usage

```
data(barto)
```

Format

barto, a list of three dataframes, representing the four genomes and their pairwise comparisons:

- dna_segswhich is a list of 4 dna_seg objects, containing all the protein genes for each genome. Obtained by reading ptt files downloaded from NCBI with read_dna_seg_from_ptt.
- comparisonswhich is a list of 3 comparison objects, obtained by doing genome-to-genome (fasta files) BLASTS, and then reading the resulting tab files with read_comparison_from_blast.
- rnt_segswhich is a list of 4 dna_seg objects, containing all the RNA genes of the four genomes. Obtained by reading rnt files downloaded from NCBI with read_dna_seg_from_ptt.

A bash script to obtain the same file as in the dataset is available in the `extdata` folder of the package. Find its location by running `system.file('extdata/barto.sh', package = 'genoPlotR')`.

References

BLAST: http://www.ncbi.nlm.nih.gov/blast/

Examples

```
data(barto)
plot_gene_map(barto$rnt_segs, barto$comparisons, gene_type="blocks")
```

`c.dna_seg` *Concatenate dna_seg objects*

Description

Concatenate dna_seg objects.

Usage

```
## S3 method for class 'dna_seg':
c(...)
```

Arguments

`...`	`dna_seg`s to be concatenated.

Value

A `dna_seg` object

Author(s)

Lionel Guy

See Also

`dna_seg`

Examples

```
## load data
data(three_genes)

dna_segs[1:2]
c(dna_segs[[1]], dna_segs[[2]])
```

chrY_subseg — *Comparisons of subsegments of the Y chromosome in human and chimp*

Description

A subsegment of the Y chromosome in Homo sapiens and Pan troglodytes, to illustrate support for exons and introns.

Usage

```
data(chrY_subseg)
```

Format

A list of two data frames, representing the Y segment in the two species, and containing:

- `dna_segs`which is a list of two `dna_seg` objects, containing each three rows (or genes).
- `comparison`which is a list of one `comparison` objects.

Details

Header for the Homo sapiens genbank file: LOCUS NC_000023 220001 bp DNA linear CON 10-JUN-2009 DEFINITION Homo sapiens chromosome X, GRCh37 primary reference assembly. ACCESSION NC_000023 REGION: 2600000..2820000 GPC_000000047

Header for the Pan troglodytes file: LOCUS NC_006491 220001 bp DNA linear CON 18-SEP-2006 DEFINITION Pan troglodytes chromosome X, reference assembly (based on Pan_troglodytes-2.1). ACCESSION NC_006491 REGION: 2620000..2840000

Examples

```
data(chrY_subseg)
plot_gene_map(chrY_subseg$dna_segs, chrY_subseg$comparison, dna_seg_scale=TRUE,
              scale=FALSE)
```

comparison — *Comparison class and class functions*

Description

A comparison is a collection of similarities, representing the comparison between two DNA segments. These functions are class functions to create, convert and test comparison objects.

Usage

```
comparison(x)
as.comparison(df)
is.comparison(comparison)
```

Arguments

x	Can be a list or data.frame object. See the details for the columns in the data.frame.
df	A data.frame object. See details for the required columns.
comparison	Any object to test.

Details

Objects (either data frames or lists) should have at least named elements start1, end1, start2 and end2. In addition, it can take a color column. Additional numeric columns can be used for color-coding (via apply_color_scheme.

comparison tries to build a comparison object from either a data frame or a list, as.comparison accepts only data.frames.

is.comparison returns TRUE if the object tested is a comparison object.

Read functions such as read_comparison_from_tab and read_comparison_from_blast also return comparison objects.

Value

A comparison object for comparison and as.comparison. Comparison objects are also of class data.frame. They contain the columns start1, end1, start2, end2, direction and col (color).

A logical for is.comparison.

Author(s)

Lionel Guy

See Also

dna_seg, read_comparison_from_tab, read_comparison_from_blast, trim.comparison, reverse.comparison.

Examples

```
## Get some values
starts1 <- c(2, 1000, 1050)
ends1 <- c(600, 800, 1345)
starts2 <- c(50, 800, 1200)
ends2 <- c(900, 1100, 1322)

## From a data.frame
comparison1 <- as.comparison(data.frame(start1=starts1, end1=ends1,
                                        start2=starts2, end2=ends2))
comparison1
is.comparison(comparison1)
is.data.frame(comparison1)
comparison(data.frame(start1=starts1, end1=ends1,
                      start2=starts2, end2=ends2))
## From a list
comparison(list(start1=starts1, end1=ends1,
                start2=starts2, end2=ends2))
```

```
## From a file
comparison2_file <- system.file('extdata/comparison2.tab',
                                package = 'genoPlotR')
comparison2 <- read_comparison_from_tab(comparison2_file)
```

---

dna_seg *DNA segment (dna_seg) class and class functions*

Description

A DNA segment is a collection of genes or elements along a genome, to be represented on a map. These functions are class functions to create, convert and test dna_seg objects.

Usage

```
dna_seg(x, ...)
as.dna_seg(df, col = "blue", lty = 1, lwd = 1, pch = 8, cex = 1, gene_type = "ar
is.dna_seg(dna_seg)
```

Arguments

x	A data.frame or list that can be coerced to a data frame.
...	Arguments further passed to as.dna_seg (see below).
df	A data frame representing the dna_seg object. See details for necessary columns.
col	Either a color vector of the same length as df or of length one, to be applied to the whole object. Default to blue.
lty	A vector of the same length as df or of length one, giving the line type around the objects.
lwd	Same as lty, giving the line width.
pch	Same as lty, giving the character representing each object. Goes with gene_type points.
cex	Same as lty, giving the character size representing each object. Goes with gene_type points.
gene_type	Vector of the same length as df or of length one, giving the type of representation of each object.
dna_seg	Object to test.

Details

Objects to be converted needs to have their first 4 columns named name, start, end and strand. Extra columns with names col, lty, lwd, pch, cex, gene_type will be used in the plotting process. Other extra columns will be kept in the object, but not used.

dna_seg tries to build a dna_seg object from a data frame or a list.

as.dna_seg tries to build a dna_seg object from a data frame.

Read functions such as read_dna_seg_from_tab and read_dna_seg_from_ptt also return dna_seg objects.

Value

A comparison object for `comparison` and `as.comparison`. DNA seg objects are also of class `data.frame`. They contain the following columns: `name`, `start`, `end`, `strand`, `col`, `lty`, `lwd`, `pch`, `cex`, `gene_type`.

A logical for `is.comparison`.

Author(s)

Lionel Guy

See Also

`read_dna_seg_from_tab`, `read_dna_seg_from_ptt`, `gene_types`.

Examples

```
## generate data
names1 <- c("feat1", "feat2", "feat3")
starts1 <- c(2, 1000, 1050)
ends1 <- c(600, 800, 1345)
strands1 <- c("-", -1, 1)
cols1 <- c("blue", "grey", "red")

## create data.frame
df1 <- data.frame(name=names1, start=starts1, end=ends1,
                  strand=strands1, col=cols1)

## with dna_seg
dna_seg1 <- dna_seg(df1)
dna_seg1
as.dna_seg(df1)

## test
is.dna_seg(dna_seg1)

## directly readable with read_dna_seg_from_tab
## Not run:
write.table(x=dna_seg1, file="dna_seg1.tab", quote=FALSE,
            row.names=FALSE, sep="\t")

## End(Not run)

## with only one gene and with list, or two, and merging with c.dna_seg
gene2a <- dna_seg(list(name="feat1", start=50, end=900, strand="-", col="blue"))
genes2b <- dna_seg(data.frame(name=c("feat2", "feat3"), start=c(800, 1200),
                              end=c(1100, 1322), strand=c("+", 1),
                              col=c("grey", "red")))
dna_seg2 <- c(gene2a, genes2b)

## test
is.dna_seg(dna_seg2)

## reading from file
dna_seg3_file <- system.file('extdata/dna_seg3.tab', package = 'genoPlotR')
dna_seg3 <- read_dna_seg_from_tab(dna_seg3_file)
is.dna_seg(dna_seg3)
```

gene_types *Gene types*

Description

Returns a vector containing the available gene types.

Usage

```
gene_types(auto = TRUE)
```

Arguments

auto	Logical. Should type "auto" be added?

Value

A character vector.

Author(s)

Lionel Guy

See Also

plot_gene_map

Examples

```
gene_types()

## Load data
data(barto)
n <- length(gene_types(auto=FALSE))

## Get a small subset from the barto dataset
dna_seg <- barto$dna_segs[[3]][1:n,]
plot_gene_map(list(dna_seg))

# Change gene_types and plot again
dna_seg$gene_type <- gene_types(auto=FALSE)
dna_seg$col <- rainbow(n)
dna_seg_r <- dna_seg
dna_seg_r$strand <- -dna_seg$strand
plot_gene_map(list(dna_seg, dna_seg_r))
```

human_nt *Human-readable nucleotide scale*

Description

Return a human readable list from a nucleotide position or lenght.

Usage

```
human_nt(nt, signif = FALSE)
```

Arguments

nt	A nucleotide position
signif	Either a logical or an integer. If FALSE (default), nt is not rounded. Else, it returns signif significant digits.

Details

Return a nucleotide value in nt, kb, Mb or Gb, according to the value given. This is particularly useful to display nice scales without too many trailing zeros.

Value

Returns a list with 4 elements

n	A numeric value corresponding to nt divided by mult (see below).
tag	A character, giving the multiplier used in text.
mult	The muliplier used, in numeric value.
text	A character, giving the value in a human readable format.

Author(s)

Lionel Guy

Examples

```
human_nt(123456)
human_nt(123456, signif=2)
human_nt(123456890, signif=2)
```

mauve_bbone *Mauve backbone of 4 Bartonella genomes*

Description

The result of a multiple genome alignment with Mauve.

Usage

```
data(mauve_bbone)
```

Format

`bbone`, a list of two dataframes, representing the regions which are conserved in at least two genomes:

- `dna_segs`which is a list of 4 `dna_seg` objects, containing the mauve blocks for each genome.
- `comparisons`which is a list of 3 `comparison` objects.

A bash script to obtain the same file as in the data is available in the `extdata` folder of the package. Find its location by running `system.file('extdata/mauve.sh', package = 'genoPlotR')`.

The resulting backone file can then be read with `read_mauve_backbone`.

References

Mauve: http://asap.ahabs.wisc.edu/mauve/

Examples

```
data(mauve_bbone)
plot_gene_map(bbone$dna_segs, bbone$comparisons)
```

middle *Middles of a dna_seg*

Description

Returns a vector containing the middle of the genes of a dna_seg. Useful to prepare annotations, for example.

Usage

```
middle(dna_seg)
```

Arguments

`dna_seg`	A `dna_seg` object.

Value

A numeric vector.

Author(s)

Lionel Guy

See Also

annotation, dna_seg

Examples

```
## Load data
data(barto)

## Get middles of the first dna_seg
mid <- middle(barto$dna_segs[[1]])
```

---

plot_gene_map *Plot gene and genome maps*

Description

This plotting function represents linearly DNA segments and their comparisons. It will plot one line per DNA segment, eventually separated by the comparisons. In addition, a tree can be plotted on the left of the plot, and annotations on the top row. Since this is a grid plot, it can be placed into other graphics, or modified subsequently.

Usage

```
plot_gene_map(dna_segs,
              comparisons = NULL,
              tree = NULL,
              tree_width = NULL,
              legend = NULL,
              annotations = NULL,
              annotation_height = 1,
              annotation_cex = 0.8,
              xlims = NULL,
              offsets = NULL,
              minimum_gap_size = 0.05,
              fixed_gap_length = FALSE,
              limit_to_longest_dna_seg = TRUE,
              main = NULL,
              main_pos = "centre",
              dna_seg_labels = NULL,
              dna_seg_label_cex=1,
              gene_type = NULL,
              arrow_head_len = 200,
```

```
                dna_seg_line = TRUE,
                scale = TRUE,
                dna_seg_scale = !scale,
                n_scale_ticks=7,
                scale_cex=0.6,
                global_color_scheme = c("auto", "auto", "blue_red"),
                override_color_schemes = FALSE,
                plot_new=TRUE,
                debug = 0
                )
```

Arguments

dna_segs	A list of dna_seg objects. Mandatory.
comparisons	A list of comparison objects. Optional.
tree	A tree, under the form of a phylog object. If specified, takes place at the left of the tags. See details below for more information.
tree_width	Numeric. The width of the tree area in the plot, in inches. By default, takes 20 percent of the total plot.
legend	Yet unimplemented.
annotations	An annotation object or a list of annotation objects. See details. Optional.
annotation_height	Numeric. The height, in lines, of the annotation line. One by default, if annotation is defined.
annotation_cex	Numeric. The cex (i.e. the character expansion) of the annotation line.
xlims	A list with as many elements as there are dna_segs, or NULL. If NULL, the whole segment will be represented. If a list, each element of the list is a numeric vector, representing pairs of left and right limits for each subsegment. See details.
offsets	A list or a vector with as many elements as there are dna_segs, or NULL. If is a numeric vector, gives the offset of the first subsegment. If is a list, each element should have the same length as there are subsegments (see xlims). Gives then the length of each gap. If NULL, the size of the gaps is optimized to minimize the lengths of the comparisons. See details.
minimum_gap_size	A numeric. How much of the plotting region should a gap be, at least. Default is 0.05 (20% of the plotting region).
fixed_gap_length	Should the gaps have a fixed length? Otherwise, the gap length will be optimized to minimize the size of comparisons. FALSE by default.
limit_to_longest_dna_seg	A logical. Should the plot be restricted to the longest dna_seg? If no, the other segments can be extended to better fit comparisons.
main	A character. Main title of the plot.
main_pos	Position of the main title. One of centre, left or right.
dna_seg_labels	A character, same length as dna_segs. The names of the segments. If NULL, the names of dna_segs will be used, if available. Else, no name are plotted. If a tree is given, names must exist either in dna_seg_labels or in the names of dna_segs.

dna_seg_label_cex
: A numeric. The character size for the DNA segments labels, or tree labels. Default is 1.

gene_type
: A character. Describes the type of representation of genes or dna_seg elements. See details.

arrow_head_len
: A numeric. Gives the length of arrow heads for gene type "arrows". The arrow head extends at maximum at half of the gene. Set to Inf to have all arrow heads covering the half of the gene. 200 by default.

dna_seg_line
: A vector, either logical or giving colors, of length 1 or of same length as dna_segs. Should the line in the middle of the segments be drawn, and if yes, in what color. TRUE by default, which gives black lines. FALSE (logical, or as a string) results in no plotting.

scale
: A logical. Should the scale be displayed on the plot. TRUE by default.

dna_seg_scale
: A logical, of length one or of the same length as dna_segs. Should a scale be displayed below each or all dna segments, respectively. !scale by default.

n_scale_ticks
: A integer. The (approximate) number of ticks on the longest segment. Default: 7.

scale_cex
: A numeric. The character size for the scale labels. Default is 1.

global_color_scheme
: A character of length 3. If no col column is present on any comparison or is override_color_schemes is set, apply a global color scheme over all comparions. See below for more details. c("auto", "auto", "blue_red") by default.

override_color_schemes
: A logical. If TRUE, apply a global color scheme even if there are comparisons that have col columns. FALSE by default.

plot_new
: Logical. Produce a new plot? If TRUE, uses grid.newpage before plotting.

debug
: A numeric. If > 0, only that number of element will be plotted for each dna_seg and comparison.

Details

One line is plotted per dna_seg. Eventually, the space between the lines will be filled with the comparisons.

A phylogenetic tree (a phylog object from package ade4) can be drawn at the left of the plot. The tree doesn't need to be ordered as the dna_seg_labels, but a permutation of the tree with that order should exist. If the tree is large, the number of permutations become too large, and the function will stop (>100000 permutations). The solution is then to provide segments that are ordered in the same manner as the tree labels (or vice-versa).

A scale, a main title and an annotation row at the top of the plot can also be added.

The format of the elements of dna_segs is previously determined in the object or can be globally set by gene_type. See the function gene_types to return the available types.

xlims allow the user to plot subsegments of a dna_seg. xlims consists of a list composed of as many numeric vectors as there are segments. Each of these numeric vectors give pairs of left and right borders, and gives the direction. For example, c(1,2,6,4) will plot two subsegments, segment

1 to 2 which is plotted left to right and segment 4 to 6, plotted right to left. `-Inf` and `Inf` values are accepted. `NULL` values will result in plotting the whole segment.

`offsets` allows to user to define the placement of the subsegments. If a list is provided, each element of the list should have as many elements as there are subsegments. It will give the size of the gaps, including the first one from the border of the plot to the first subsegment.

`dna_seg_scale` gives the ability to plot scales on one, some or every segment. `c(TRUE, FALSE, TRUE)` will add scales to the first and third segments.

The three elements of `global_color_scheme` are (i) which column serves as scale to apply the color scheme, or "auto" (default); (ii) if the scale is "increasing" or "decreasing" (see `apply_color_scheme` for more details), or "auto" (default); (iii) the color scheme to apply.

Value

Nothing. A lattice graphic is plotted on the current device.

Note

This plotting function has been tested as far as possible, but given its complexity and that the package is young, bugs or strange behaviors are possible. Please report them to the author.

As of 10/3/2010, support for viewing exons/introns is only available using genbank and embl formats, not when importing ptt files.

Author(s)

Lionel Guy `<lionel.guy@ebc.uu.se>`, Jens Roat Kultima

See Also

`dna_seg` and `comparison` for the base objects; `read_dna_seg_from_tab`, `read_dna_seg_from_ptt`, `read_comparison_from_tab` and `read_comparison_from_blast` to read from files; `gene_types` for `gene_type` argument; `apply_color_scheme` for color schemes;

Examples

```
old.par <- par(no.readonly=TRUE)
data("three_genes")

## Segments only
plot_gene_map(dna_segs=dna_segs)

## With comparisons
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons)

## Tree
names <- c("A_aaa", "B_bbb", "C_ccc")
names(dna_segs) <- names
tree <- newick2phylog("(((A_aaa:4.2,B_bbb:3.9):3.1,C_ccc:7.3):1);")
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              tree=tree)

## Increasing tree width
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              tree=tree, tree_width=3)
```

```
## Annotation
## Calculating middle positions
mid_pos <- middle(dna_segs[[1]])

# Create first annotation
annot1 <- annotation(x1=mid_pos, text=dna_segs[[1]]$name)
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons, annotations=annot1)

## Exploring options
annot2 <- annotation(x1=c(mid_pos[1], dna_segs[[1]]$end[2]),
                     x2=c(NA, dna_segs[[1]]$end[3]),
                     text=c(dna_segs[[1]]$name[1], "region1"),
                     rot=c(30, 0), col=c("grey", "black"))
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              annotations=annot2, annotation_height=1.3)

## xlims
## Just returning a segment
plot_gene_map(dna_segs, comparisons,
              xlims=list(NULL, NULL, c(Inf,-Inf)),
              dna_seg_scale=TRUE)
## Removing one gene
plot_gene_map(dna_segs, comparisons,
              xlims=list(NULL, NULL, c(-Inf,2800)),
              dna_seg_scale=TRUE)

## offsets
offsets <- c(0, 0, 0)
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons, offsets=offsets)
offsets <- c(200, 400, 0)
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons, offsets=offsets)

## main
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              main="Comparison of A, B and C")
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              main="Comparison of A, B and C", main_pos="left")

## dna_seg_labels
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              dna_seg_labels=c("Huey", "Dewey", "Louie"))

## dna_seg_labels size
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              dna_seg_labels=c("Huey", "Dewey", "Louie"),
              dna_seg_label_cex=2)

## dna_seg_line
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              dna_seg_line=c("FALSE", "red", grey(0.6)))

## gene_type
plot_gene_map(dna_segs=dna_segs, comparisons=comparisons,
              gene_type="side_blocks")

##
## From here on, using a bigger dataset from a 4-genome comparison
```

```
##
data("barto")
## Adding a tree
tree <- newick2phylog("(BB:2.5,(BG:1.8,(BH:1,BQ:0.8):1.9):3);")
## Showing only subsegments
xlims1 <- list(c(1380000, 1445000),
               c(10000, 83000),
               c(15000, 98000),
               c(5000, 82000))
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              xlims=xlims1,
              dna_seg_scale=TRUE)
## Showing several subsegments per genome
xlims2 <- list(c(1445000, 1415000, 1380000, 1412000),
               c(  10000,   45000,   50000,   83000, 90000, 120000),
               c(  15000,   36000,   90000,  120000, 74000,  98000),
               c(   5000,    82000))
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              xlims=xlims2,
              dna_seg_scale=TRUE)

## dna_seg_scale & global_color_scheme
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree, xlims=xlims1,
              dna_seg_scale=c(TRUE, FALSE, FALSE, TRUE), scale=FALSE,
              global_color_scheme=c("e_value","auto","grey"))

## Changing size and number of dna_seg_scale_labels
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree, xlims=xlims1,
              dna_seg_scale=TRUE, scale=FALSE,
              n_scale_ticks=3, scale_cex=1)

## Changing size of dna_seg_labels (tree, in that case)
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree, xlims=xlims1,
              dna_seg_scale=TRUE, scale=FALSE,
              dna_seg_label_cex=1.7)


## Adding annotations for all genomes
annots <- lapply(barto$dna_segs, function(x){
  mid <- middle(x)
  annot <- annotation(x1=mid, text=x$name, rot=30)
  # removing gene names starting with "B" and keeping 1 in 4
  idx <- grep("^[^B]", annot$text, perl=TRUE)
  annot[idx[idx %% 4 == 0],]
})
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              annotations=annots,
              xlims=xlims2,
              dna_seg_scale=TRUE)
## Allow segments to be placed out of the longest segment
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              annotations=annots,
              xlims=xlims2,
              limit_to_longest_dna_seg=FALSE,
              dna_seg_scale=TRUE)
## Hand-made offsets: only placement of first segment
offsets1 <- c(10000, 0, 30000, 10000)
```

```
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              annotations=annots,
              xlims=xlims2,
              offsets=offsets1,
              dna_seg_scale=TRUE)
## Hand-made offsets: size of all gaps
offsets2 <- list(c(10000, 10000),
                 c(2000, 2000, 2000),
                 c(10000, 5000, 2000),
                 c(10000))
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              annotations=annots,
              xlims=xlims2,
              offsets=offsets2,
              dna_seg_scale=TRUE)

##
## Exploring and modifying a gene map plot
##
plot_gene_map(barto$dna_segs, barto$comparisons, tree=tree,
              annotations=annots,
              xlims=xlims2,
              offsets=offsets2,
              dna_seg_scale=TRUE)
## View viewports
current.vpTree()
## Go down to one of the viewports, add an xaxis, go back up to root viewport
downViewport("dna_seg_scale.3.2")
grid.rect()
upViewport(0)
## Get all the names of the objects
grobNames <- getNames()
grobNames
## Change the color ot the scale line
grid.edit("scale.lines", gp=gpar(col="grey"))
## Remove first dna_seg_lines
grid.remove("dna_seg_line.1.1")

##
## Plot genoPlotR logo
##
col <- c("#B2182B", "#D6604D", "#F4A582", "#FDDBC7",
         "#D1E5F0", "#92C5DE", "#4393C3", "#2166AC")
cex <- 2.3
## First segment
start1 <- c(150, 390, 570)
end1   <- c(  1, 490, 690)
genoR <- c(270, 530)
## Second segment
start2 <- c(100, 520, 550)
end2   <- c(240, 420, 650)
Plot <- c(330)
## dna_segs
ds1 <- as.dna_seg(data.frame(name=c("", "", ""),
                             start=start1, end=end1, strand=rep(1, 3),
                             col=col[c(2, 6, 1)], stringsAsFactor=FALSE))
ds_genoR <- as.dna_seg(data.frame(name=c("geno", "R"),
```

```
                                start=genoR, end=genoR, strand=rep(1, 2),
                                col=c(col[8], "black"),
                                stringsAsFactor=FALSE), cex=cex, gene_type="text")
ds2 <- as.dna_seg(data.frame(name=c("", "", ""),
                                start=start2, end=end2, strand=rep(1, 3),
                                col=col[c(5, 3, 7)],
                                stringsAsFactor=FALSE))
ds_Plot <- as.dna_seg(data.frame(name="Plot",
                                start=Plot, end=Plot, strand=1,
                                col=col[c(1)],
                                stringsAsFactor=FALSE), cex=cex, gene_type="text")
## comparison
c1 <- as.comparison(data.frame(start1=start1, end1=end1,
                                  start2=start2, end2=end2,
                                  col=grey(c(0.6, 0.8, 0.5))))
## Generate genoPlotR logo
## Not run:
cairo_pdf("logo.pdf", h=0.7, w=3)

## End(Not run)
par(fin=c(0.7, 3))
plot_gene_map(dna_segs=list(c(ds1, ds_genoR), c(ds2, ds_Plot)),
              comparisons=list(c1), scale=FALSE, dna_seg_scale=FALSE,
              dna_seg_line=grey(0.7), offsets=c(-20,160))
## Not run:
dev.off()

## End(Not run)
par(old.par)
```

`range.dna_seg` *Range calculation*

Description

Calculate the range of dna_seg and comparisons.

Usage

```
## S3 method for class 'dna_seg':
range(x, ...)
## S3 method for class 'comparison':
range(x, overall=TRUE, ...)
## S3 method for class 'annotation':
range(x, ...)
```

Arguments

`x`	Object to calculate the range from.
`overall`	Logical, `TRUE` by default. Should the range be calculated over the whole object? If `FALSE`, a range is calculated on each side of the comparison.
`...`	Unused.

Details

Calculate the overall range of a dna_seg, comparison or an annotation object.

Value

A numeric of length 2. For comparison, if overall is FALSE, a data frame with two rows and two columns, xlim1 and xlim2.

Author(s)

Lionel Guy

See Also

dna_seg, comparison, trim for further examples.

Examples

```
## Load data
data(three_genes)

## On dna_seg
dna_segs[[1]]
range(dna_segs[[1]])

## On comparison
comparisons[[2]]
range(comparisons[[2]])
range(comparisons[[2]], overall=FALSE)
```

---

read_functions *Reading functions*

Description

Functions to parse dna_seg objects from tab, embl, genbank, fasta, ptt files or from mauve backbone files, and comparison objects from tab or blast files.

Usage

```
read_dna_seg_from_tab(file, header = TRUE, ...)
read_dna_seg_from_file(file, tagsToParse=c("CDS"), fileType = "detect",
                       meta_lines = 2, gene_type = "auto", header = TRUE, ...)
read_dna_seg_from_embl(file, tagsToParse=c("CDS"), ...)
read_dna_seg_from_genbank(file, tagsToParse=c("CDS"), ...)
read_dna_seg_from_fasta(file, ...)
read_dna_seg_from_ptt(file, meta_lines = 2, header = TRUE, ...)
read_comparison_from_tab(file, header = TRUE, ...)
read_comparison_from_blast(file, sort_by = "per_id",
                           filt_high_evalue = NULL,
                           filt_low_per_id = NULL,
                           filt_length = NULL,
```

```
                                color_scheme = NULL, ...)
read_mauve_backbone(file, ref = 1, gene_type = "side_blocks",
                    header = TRUE, filter_low = 0, ...)
```

Arguments

file	Path to file to load. URL are accepted.
header	Logical. Does the tab file has headers (column names)?
tagsToParse	Character vector. Tags to parse in embl or genbank files. Common tags are 'CDS', 'gene', 'misc_feature'.
fileType	Character string. Select file type, could be 'detect' for automatic detection, 'embl' for embl files, 'genbank' for genbank files or 'ptt' for ptt files.
meta_lines	The number of lines in the ptt file that represent "meta" data, not counting the header lines. Standard for NCBI files is 2 (name and length, number of proteins. Default is also 2.
gene_type	Determines how genes are visualized. If 'auto' genes will appear as arrows in there are no introns and as blocks if there are introns. Can also be set to for example 'blocks' or 'arrows'. Do note, currently introns are not supported in the ptt file format. Default for mauve backbone is side_blocks. See gene_types page for more details, or use function gene_types.
sort_by	In BLAST-like tabs, gives the name of the column that will be used to sort the comparisons. Accepted values are per_id (percent identity, default), mism (mismatches), gaps (gaps), e_value (E-value), bit_score (bit score).
filt_high_evalue	A numerical, or NULL (default). Filters out all comparisons that have a e-value higher than this one.
filt_low_per_id	A numerical, or NULL (default). Filters out all comparisons that have a percent identity lower than this one.
filt_length	A numerical, or NULL (default). Filters out all comparisons that have alignments shorter than this value.
color_scheme	A color scheme to apply. See apply_color_scheme for more details. Possible values include grey and red_blue. NULL by default. Color schemes can be applied while running plot_gene_map.
ref	In mauve backbone, which of the dna segments will be the reference, i.e. which one will have its blocks in order.
...	Further arguments passed to generic reading functions and class conversion functions. See as.dna_seg and as.comparison. For read_comparison* functions, see details.
filter_low	A numeric. If larger than 0, all blocks smaller that this number will be filtered out. Defaults to 0.

Details

Tab files representing DNA segements should have at least the following columns: name, start, end, strand (in that order. Additionally, if the tab file has headers, more columns will be used to define, for example, the color, line width and type, pch and/or cex. See dna_seg for more information. An example:

name	start	end	strand	col

feat1A	2	1345	1	blue
feat1B	1399	2034	1	red
feat1C	2101	2932	-1	grey
feat1D	2800	3120	1	green

Embl and Genbank files are two commonly used file types. These file types often contain a great variety of information. To properly extract data from these files, the user has to choose which features to extract. Commonly 'CDS' features are of interest, but other feature tags such as 'gene' or 'misc_feature' may be of interest. Should a feature contain an inner "pseudo" tag indicating this CDS or gene is a pseudo gene, this will be presented as a 'CDS_pseudo' or a 'gene_pseudo' feature type respectively in the resulting table. Certain constraints apply to these file types, of which some are: embl files must contain one and only one ID tag; genbank files may only contain one and only one locus tag.

Fasta files are read as one gene, as long as there are nucleotides in the fasta file.

Ptt (or protein table) files are a tabular format giving a bunch of information on each protein of a genome (or plasmid, or virus, etc). They are available for each published genome on the NCBI ftp site (ftp://ftp.ncbi.nlm.nih.gov/genomes/). As an example, look at ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Bartonella
1/NC_005956.ptt.

Tabular comparison files should have at least the following columns: start1, end1, start2, end2. If no header is specified, the fifth column is parsed as the color.

start1	end1	start2	end2	col
2	1345	10	1210	red
1399	2034	2700	1100	blue
500	800	3000	2500	blue

BLAST tabular result files are produced either with blastall using -m8 or -m9 parameter, or with any of the newer blastn/blastp/blastx/tblastx using -outfmt 6 or -outfmt 7.

In the subsequent `plot_gene_map`, the comparisons are drawn in the order of the `comparison` object, i.e. the last rows of the comparison object are on the top in the plot. For comparisons read from BLAST output, the order can be modified by using the argument `sort_by`. In any case, the order of plotting can be modified by modifying the order of rows in the `comparison` object prior to plotting.

Mauve backbone is another tabular data file that summarizes the blocks that are similar between all compared genomes. Each genome gets two columns, one start and one end of the block. There is one row per block and eventually a header row. If named, columns have sequence numbers, not actual names, so be careful to input the same order in both Mauve and genoPlotR. See http://asap.ahabs.wisc.edu/mauve-aligner/mauve-user-guide/mauve-output-file-formats.html for more info on the file format. Normally, the function should be able to read both `progressiveMauve` and `mauveAligner` outputs. The function returns both the blocks as `dna_seg`s and the links between the blocks as `comparison`s.

Value

`read_dna_seg_from_tab`, `read_dna_seg_from_file`, `read_dna_seg_from_embl`, `read_dna_seg_from_genbank` and `read_dna_seg_from_ptt` return `dna_seg` objects. `read_comparison_from_tab` and `read_comparison_from_blast` return `comparison` objects. `read_mauve_backbone` returns a list containing a list of `dna_seg`s and `comparison`s. objects.

Note

Formats are changing and it maybe that some functions are temporarily malfunctioning. Please report any bug to the author. Mauve examples were prepared with Mauve 2.3.1.

Author(s)

Lionel Guy, Jens Roat Kultima

References

For BLAST: http://www.ncbi.nlm.nih.gov/blast/ For Mauve: http://asap.ahabs.wisc.edu/mauve/

See Also

comparison, dna_seg, apply_color_scheme.

Examples

```
##
## From tabs
##
## Read DNA segment from tab
dna_seg3_file <- system.file('extdata/dna_seg3.tab', package = 'genoPlotR')
dna_seg3 <- read_dna_seg_from_tab(dna_seg3_file)

## Read comparison from tab
comparison2_file <- system.file('extdata/comparison2.tab',
                                package = 'genoPlotR')
comparison2 <- read_comparison_from_tab(comparison2_file)

##
## Mauve backbone
##
## File
bbone_file <- system.file('extdata/barto.backbone', package = 'genoPlotR')
## Read backbone
bbone <- read_mauve_backbone(bbone_file)
names <- c("B_bacilliformis", "B_grahamii", "B_henselae", "B_quintana")
names(bbone$dna_segs) <- names
## Plot
plot_gene_map(dna_segs=bbone$dna_segs, comparisons=bbone$comparisons)

## Using filter_low & changing reference sequence
bbone <- read_mauve_backbone(bbone_file, ref=2, filter_low=2000)
names(bbone$dna_segs) <- names
plot_gene_map(dna_segs=bbone$dna_segs, comparisons=bbone$comparisons)

## Read guide tree
tree_file <- system.file('extdata/barto.guide_tree', package = 'genoPlotR')
tree_str <- readLines(tree_file)
for (i in 1:length(names)){
  tree_str <- gsub(paste("seq", i, sep=""), names[i], tree_str)
}
tree <- newick2phylog(tree_str)
## Plot
plot_gene_map(dna_segs=bbone$dna_segs, comparisons=bbone$comparisons,
```

```
              tree=tree)

##
## From embl file
##
bq_embl_file <- system.file('extdata/BG_plasmid.embl', package = 'genoPlotR')
bq <- read_dna_seg_from_embl(bq_embl_file)

##
## From genbank file
##
bq_genbank_file <- system.file('extdata/BG_plasmid.gbk', package = 'genoPlotR')
bq <- read_dna_seg_from_file(bq_genbank_file, fileType="detect")

##
## From ptt files
##
## From a file
bq_ptt_file <- system.file('extdata/BQ.ptt', package = 'genoPlotR')
bq <- read_dna_seg_from_ptt(bq_ptt_file)
## Read directly from NCBI ftp site:
url <- "ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Bartonella_henselae_Houston-1/NC_005956.p
attempt <- 0
## Not run:
while (attempt < 5){
  attempt <- attempt + 1
  bh <- try(read_dna_seg_from_ptt(url))
  if (!inherits(bh, "try-error")) {
    attempt <- 99
  } else {
    print(paste("Tried", attempt, "times, retrying in 5s"))
    Sys.sleep(5)
  }
}

## End(Not run)
## If attempt to connect to internet fails
if (!exists("bh")){
  data(barto)
  bh <- barto$dna_segs[[3]]
}

##
## Read from blast
##
bh_vs_bq_file <- system.file('extdata/BH_vs_BQ.blastn.tab',
                             package = 'genoPlotR')
bh_vs_bq <- read_comparison_from_blast(bh_vs_bq_file, color_scheme="grey")

## Plot
plot_gene_map(dna_segs=list(BH=bh, BQ=bq), comparisons=list(bh_vs_bq),
              xlims=list(c(1,50000), c(1, 50000)))
```

reverse *Reverse objects*

Description

Reverse objects, mainly dna_seg and comparison

Usage

```
reverse(x, ...)
## Default S3 method:
reverse(x, ...)
## S3 method for class 'dna_seg':
reverse(x, ...)
## S3 method for class 'comparison':
reverse(x, side = 0, ...)
```

Arguments

x	The object to reverse.
...	Unused.
side	In the case of comparisons, the side of the comparison that should be reversed. If side=1, the first side will be reversed. If side=2, the second side will be reversed. If side<1, no side is reversed. If side>2, both sides are reversed.

Value

The same object as input.

Author(s)

Lionel Guy

See Also

dna_seg, comparison

Examples

```
## load data
data(three_genes)

## on dna_seg
dna_segs[[1]]
reverse(dna_segs[[1]])
## on comparison
reverse(comparisons[[2]], side=1)
reverse(comparisons[[2]], side=3)

## With mauve backbone
data(mauve_bbone)
## Plot
```

```
plot_gene_map(dna_segs=bbone$dna_segs, comparisons=bbone$comparisons)

## Reverse B_bacilliformis, and the corresponding comparison (first "side")
bbone$dna_segs[[1]] <- reverse(bbone$dna_segs[[1]])
bbone$comparisons[[1]] <- reverse(bbone$comparisons[[1]], 1)
plot_gene_map(dna_segs=bbone$dna_segs, comparisons=bbone$comparisons)
```

---

three_genes *Three genes data set*

Description

A set of three made-up genes, compared in three chromosomes.

Usage

```
data(three_genes)
```

Format

Two dataframes, representing the three genes in three DNA segments:

- `dna_segs`which is a list of three `dna_seg` objects, containing each three rows (or genes).
- `comparisons`which is a list of two `comparison` objects.

Examples

```
data(three_genes)
plot_gene_map(dna_segs, comparisons)
```

---

trim *Trimming data frames with >= 2 numeric columns*

Description

Trims data frames with 2 or more numeric columns using a xlim. `xlim`(s) are as used to filter rows whose numeric values are included in this interval.

Usage

```
trim(x, ...)
## Default S3 method:
trim(x, xlim = NULL, ...)
## S3 method for class 'dna_seg':
trim(x, xlim = NULL, ...)
## S3 method for class 'comparison':
trim(x, xlim1 = c(-Inf, Inf), xlim2 = c(-Inf, Inf), ...)
## S3 method for class 'annotation':
trim(x, xlim = NULL, ...)
```

Arguments

x	An object to trim,. generally a data frame or a matrix.
xlim	A numeric of length 2. In a general case, the rows whose values are included in this interval are returned.
...	Unused.
xlim1	A numeric of length 2. In the case of comparison, where the comparison can be filtered on two sides, the interval to filter the first side.
xlim2	A numeric of length 2. The interval to filter the second side.

Value

Returns the same object as input, with the rows corresponding to the interval given.

Author(s)

Lionel Guy

See Also

dna_seg, comparison

Examples

```
## Load
data(barto)
xlim_ref <- c(10000, 45000)
## Seg 2 (ref)
barto$dna_segs[[2]] <- trim(barto$dna_segs[[2]], xlim=xlim_ref)
## Seg 1
barto$comparisons[[1]] <- trim(barto$comparisons[[1]], xlim2=xlim_ref)
xlim1 <- range(barto$comparisons[[1]], overall=FALSE)$xlim1
barto$dna_segs[[1]] <- trim(barto$dna_segs[[1]], xlim=xlim1)
## Seg 3
barto$comparisons[[2]] <- trim(barto$comparisons[[2]], xlim1=xlim_ref)
xlim3 <- range(barto$comparisons[[2]], overall=FALSE)$xlim2
barto$dna_segs[[3]] <- trim(barto$dna_segs[[3]], xlim=xlim3)
## Seg 4
barto$comparisons[[3]] <- trim(barto$comparisons[[3]], xlim1=xlim3)
xlim4 <- range(barto$comparisons[[3]], overall=FALSE)$xlim2
barto$dna_segs[[4]] <- trim(barto$dna_segs[[4]], xlim=xlim4)
## Plot
plot_gene_map(barto$dna_segs, barto$comparisons)
```

Index