Package 'FindMyFriends'

April 11, 2018

Type Package

Title Microbial Comparative Genomics in R

Version 1.8.0

Date 2016-10-06

Author Thomas Lin Pedersen

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description A framework for doing microbial comparative genomics in R. The main purpose of the package is assisting in the creation of pangenome matrices where genes from related organisms are grouped by similarity, as well as the analysis of these data. FindMyFriends provides many novel approaches to doing pangenome analysis and supports a gene grouping algorithm that scales linearly, thus making the creation of huge pangenomes feasible.

URL https://github.com/thomasp85/FindMyFriends

BugReports https://github.com/thomasp85/FindMyFriends/issues

License GPL (>=2)

biocViews ComparativeGenomics, Clustering, DataRepresentation, GenomicVariation, SequenceMatching, GraphAndNetwork

VignetteBuilder knitr

LinkingTo Rcpp

Imports methods, BiocGenerics, Biobase, tools, dplyr, IRanges, Biostrings, S4Vectors, kebabs, igraph, Matrix, digest, filehash, Rcpp, ggplot2, gtable, grid, reshape2, ggdendro, BiocParallel, utils, stats

Suggests BiocStyle, testthat, knitr, rmarkdown, reutils

Collate 'FindMyFriends-package.R' 'RcppExports.R' 'aaa.R' 'generics.R' 'pgVirtual.R' 'pgVirtualLoc.R' 'pgInMemLoc.R' 'pgInMem.R' 'pgLM.R' 'pgLMLoc.R' 'pgFull.R' 'pgFullLoc.R' 'constructor.R' 'ggGraph.R' 'grouping.R' 'investigating.R' 'linearKernel.R' 'linking.R' 'modifying.R' 'pgSlim.R' 'pgSlimLoc.R' 'progress.R' 'similarities.R' 'splitting.R'

NeedsCompilation yes

RoxygenNote 5.0.1

R topics documented:

`FindMyFriends-package` *FindMyFriends: Comparative microbial genomics in R*

Description

FindMyFriends: Comparative microbial genomics in R

Details

This package has two objectives: Define a framework for working with pangenomic data in R and provide speed and memory effecient algorithms that makes it possible to create huge pangenomes in a reasonable amount of time. While providing novel algorithms itself it also makes it possible to import results from other algorithms into the framework thus facilitating doing post-processing of results from other tools that only provides an initial grouping of genes.

In order to balance speed and memory consumption FindMyFriends provides two different sequence storage modes - either in-memory or as a reference to the original fasta file. The former excels in lookup speed but can end up too unwieldy for big pangenomes with Gb of sequence data. The latter in contrast can handle extremely huge sets of genes but can in turn slow down calculations due to longer sequence lookups.

The novelty of the FindMyFriends algorithms lie primarily in the fact that they utilise allignment-free sequence comparisons based on cosine similarity of kmer feature vectors. This is substantially faster than BLAST while retaining the needed resolution. Another novelty is the introduction of Guided Pairwise Comparison - a different approach than standard all-vs-all comparisons.

Author(s)

Thomas Lin Pedersen

`.fillDefaults` *Assign object defaults to missing values*

Description

This function takes care of investigating the enclosing functions arguments and identifying the missing ones. If they are missing and a default is given this value is assigned to the enclosing functions environment

Usage

```
.fillDefaults(def)
```

Arguments

`def`	A named list of default values

Value

This function is called for its side effects

See Also

Set and get pangenome defaults with `defaults`

Examples

```
# Should only be called within methods/functions

# This will obviously fail
## Not run:
  t <- function(x) {
    x+1
  }
  t()

## End(Not run)

# Using .fillDefaults
t <- function(x, defs) {
  .fillDefaults(defs)
  x+1
}

# With defaults
t(defs=list(x=5))

# Direct setting takes precedence
t(x=2, defs=list(x=5))

# Still fails if defs doesn't contain the needed parameter
## Not run:
  t(defs=list(y='no no'))
```

```
## End(Not run)

# Usually defs are derived from the object in a method:
## Not run:
  setMethod('fillDefExample', 'pgFull',
    function(object, x, y) {
      .fillDefaults(defaults(object))
      x+y
    }
  )

## End(Not run)
```

`.loadPgExample` *Load an example pangenome*

Description

This function loads an example pangenome at various stages of calculation, useful for examples and tests.

Usage

```
.loadPgExample(lowMem = FALSE, geneLoc = FALSE, withGroups = FALSE,
  withNeighborhoodSplit = FALSE, withParalogues = FALSE)
```

Arguments

`lowMem`	logical. Should the returned object inherit from pgLM
`geneLoc`	logical. Should the returned object inherit from pgVirtualLoc
`withGroups`	logical. Should gene groups be defined
`withNeighborhoodSplit`	logical. Should neighborhoodsplitting have been performed
`withParalogues`	logical. Should paralogue linking have been performed

Value

A pgVirtual subclass object to the specifications defined

Examples

```
# Load standard (pgFull)
.loadPgExample()

# Use pgLM
.loadPgExample(lowMem=TRUE)

# Create with pgVirtualLoc subclass (here pgFullLoc)
.loadPgExample(geneLoc=TRUE)

# Create with grouping information
```

```
.loadPgExample(withGroups=TRUE)

# Create with gene groups split by neighborhood (pgVirtualLoc implied)
.loadPgExample(withNeighborhoodSplit=TRUE)

# Create with paralogue links
.loadPgExample(withGroups=TRUE, withParalogues=TRUE)
```

---

addGenomes *Add new organisms to an existing pangenome*

Description

This method allows new genomes to be added to an already processed pangenome, preserving existing grouping and adding new genes to their relevant groups. This makes it possible to gradually grow the pangenome as new sequences becomes available without redoing the grouping at each time, loosing the gene group metadata.

Usage

```
addGenomes(object, newSet, ...)

## S4 method for signature 'pgVirtual,pgVirtual'
addGenomes(object, newSet, kmerSize, lowerLimit,
  pParam, nsParam = list(), klParam = list())
```

Arguments

object	A pgVirtual subclass to merge the new genomes into
newSet	An object of the same class as object containing the new organisms to add. Grouping of the genes contained in this object can already exist, if not it will be done automatically.
...	parameters passed on.
kmerSize	The size of the kmers to use for comparing new genes to existing
lowerLimit	The lower threshold for sequence similarity, below which it is set to 0
pParam	A BiocParallelParam object
nsParam	A list of parameters to pass to neighborhoodSplit or FALSE to skip neighborhood splitting altogether. If object has had neighborhood splitting performed and nsParam is set to FALSE it is bound to cause problems, so don't do that.
klParam	A list of parameters to pass to kmerLink or FALSE to skip paralogue linking altogether. Independent of the value of klParam kmerLink will only be run if paralogue links have been defined on object beforehand.

Value

An object of the same class as object containing the new organisms from newSet and possible new gene groups from genes with no orthologues in the original pangenome.

Methods (by class)

- `object = pgVirtual,newSet = pgVirtual`: Genome addition for all pgVirtual subclasses

Examples

```
# Get base pangenome
pg <- .loadPgExample(geneLoc = TRUE, withGroups = TRUE,
                     withNeighborhoodSplit = TRUE)
# Get some additional genomes
location <- tempdir()
unzip(system.file('extdata', 'Mycoplasma.zip', package = 'FindMyFriends'),
      exdir = location)
genomeFiles <- list.files(location, full.names = TRUE, pattern = '*.fasta')[6:10]
pg2 <- pangenome(genomeFiles, translated = TRUE, geneLocation = 'prodigal')

# Combine the two (too computational heavy to include)
## Not run:
pg3 <- addGenomes(pg, pg2, nsParam = list(lowerLimit = 0.8))

## End(Not run)
```

---

addGroupInfo *Safely add group info*

Description

This method allows for adding of group metadata by specifying the name of the metadata and the gene groups it should be added to. It protects the user from overwriting information that is derived from the data, and ensures the proper formatting. Should be prefered to `groupInfo<-` for all but the simplest cases.

Usage

```
addGroupInfo(object, ...)

## S4 method for signature 'pgVirtual'
addGroupInfo(object, info, key)
```

Arguments

`object`	A pgVirtual subclass
`...`	parameters passed on.
`info`	A data.frame with information to add
`key`	Either an integer vector with the index of each gene group the rows in info corresponds to, or the name of the column in info that holds the indexes.

Value

An object of the same class as object with the new gene group information.

Methods (by class)

- pgVirtual: Add gene group info safely for all pgVirtual subclasses

See Also

Other Metadata: addOrgInfo, groupInfo, orgInfo

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Create some info
info <- data.frame(nickname=c('Tessie', 'Johnny'), index=c(4, 500))

# Add it to the object
testPG <- addGroupInfo(testPG, info=info, key='index')
```

---

addOrgInfo *Safely add organisms info*

Description

This method allows for adding of organism metadata by specifying the name of the metadata and the organisms it should be added to. It protects the user from overwriting information that is derived from the data and ensures proper formatting. Should be prefered to orgInfo<- for all but the simplest cases.

Usage

```
addOrgInfo(object, ...)

## S4 method for signature 'pgVirtual'
addOrgInfo(object, info, key)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on.
info	A data.frame with information to add
key	Either an integer vector with the index of each organism the rows in info corresponds to, or the name of the column in info that holds the indexes.

Value

An object of the same class as object with the added organism information.

Methods (by class)

- pgVirtual: Add organism info safely for all pgVirtual subclasses

See Also

Other Metadata: addGroupInfo, groupInfo, orgInfo

Examples

```
testPG <- .loadPgExample()

# Create some information
info <- data.frame(location=c('Copenhagen', 'Paris', 'London'),
    name=c('AE017243', 'AP012303', 'AE017244')
)

# Add the information
testPG <- addOrgInfo(testPG, info=info, key='name')
```

---

cdhitGrouping *Gene grouping by preclustering with CD-HIT*

Description

This grouping algorithm partly mimicks the approach used by Roary, but instead of using BLAST in the second pass it uses cosine similarity of kmer feature vectors, thus providing an even greater speedup. The algorithm uses the CD-HIT algorithm to precluster highly similar sequences and then groups these clusters by extracting a representative and clustering these using the standard FindMyFriends kmer cosine similarity.

Usage

```
cdhitGrouping(object, ...)

## S4 method for signature 'pgVirtual'
cdhitGrouping(object, kmerSize, lowerLimit, maxLengthDif,
  geneChunkSize, cdhitOpts, cdhitIter = TRUE, nrep = 1, from = 0.9,
  by = 0.05)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on.
kmerSize	The size of the kmer's used for the comparison. If two values are given the first will be used for the CD-HIT algorithm and the second will be used for the cosine similarity calculations.
lowerLimit	A numeric giving the lower bounds of similarity below which it will be set to zero.
maxLengthDif	The maximum deviation in sequence length to allow during preclustering with CD-HIT. Below 1 it describes a percentage. Above 1 it describes a fixed length.
geneChunkSize	The maximum number of genes to pass to the CD-HIT algorithm. If object contains more genes than this, CD-HIT will be run in chunks and combined with a second CD-HIT pass before the final cosine similarity grouping.

cdhitOpts	Additional arguments passed on to CD-HIT. It should be a named list with names corresponding to the arguments expected in the CD-HIT algorithm (without the dash). i, n and s/S will be overwritten based on the other parameters given to this function and all values in cdhitOpts will be converted to character using as.character
cdhitIter	Logical. Should the preclustered groups be grouped by gradually lowering the threshold in CD-Hit or by directly calculating kmer similarities between all preclusters and group by that. Defaults to TRUE
nrep	If `cdhitIter = TRUE`, controls how many iterations should be performed at each threshold level. Defaults to 1.
from	The start similarity threshold to use for the iterative CD-Hit grouping. Together with by and nrep it defines the number of times and levels CD-Hit is run. Defaults to 0.9
by	The step size to use for the iterative CD-Hit grouping. Defaults to 0.05

Value

An object of the same class as 'object'.

Methods (by class)

- `pgVirtual`: Grouping using cdhit for all pgVirtual subclasses

References

Page, A. J., Cummins, C. A., Hunt, M., Wong, V. K., Reuter, S., Holden, M. T. G., et al. (2015). Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, btv421.

Fu, L., Niu, B., Zhu, Z., Wu, S., Li, W. (2012). CD-HIT: accelerated for clustering the next generation sequencing data. *Bioinformatics*, **28** (23), 3150–3152.

Li, W. and Godzik, A. (2006) Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**, 1658–9.

See Also

Other grouping algorithms: `gpcGrouping`, `graphGrouping`, `manualGrouping`

Examples

```
testPG <- .loadPgExample()

testPG <- cdhitGrouping(testPG)
```

collapseParalogues *Merge paralogue gene groups into new gene groups*

Description

This method allows for merging of paralogue gene groups defined using kmerLink into new, bigger, gene groups.

Usage

```
collapseParalogues(object, ...)

## S4 method for signature 'pgVirtual'
collapseParalogues(object, combineInfo = "merge", ...)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on to metadata collapse function. For combineInfo='merge' sep specifies the separator - sep='none' collapses information into list elements instead of strings. For combineInfo='largest' no addition arguments are given.
combineInfo	The approach used to combine metadata from the collapsed groups. Either 'merge' for merging, 'largest' for picking information from the largest group, or a function that takes a data.frame of multiple rows and converts it to a data.frame with one row and the same columns.

Value

An object of the same class as object with the new grouping.

Methods (by class)

- pgVirtual: Merge paralogue gene groups for all pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE, withParalogues=TRUE)

# Number of gene groups before collapse
nGeneGroups(testPG)

# Number of gene groups after collapse
testPG <- collapseParalogues(testPG, combineInfo='largest')
nGeneGroups(testPG)
```

`defaults` *Access default values for a pgVirtual subclass object*

Description

This method lets the user view and set the default values used for the different algorithms in FindMyFriends. Many of the parameters are reoccuring and it can become laborious to type them in at each step. These functionalities makes it easy to set defaults on a per-pangenome basis.

Usage

```
defaults(object)

defaults(object) <- value

## S4 method for signature 'pgVirtual'
defaults(object)

## S4 replacement method for signature 'pgVirtual'
defaults(object) <- value
```

Arguments

`object`	A pgVirtual subclass
`value`	The new values to set

Details

Currently the following methods support reading defaults from a pgVirtual object. Note that only directly named arguments are supported - arguments passed on through the . . . -mechanism are not supported unless they are passed to a function that support it.

- `graphGrouping`
- `gpcGrouping`
- `variableRegions`
- `plotGroup`
- `kmerLink`
- `plotSimilarity`
- `plotTree`
- `kmerSimilarity`

Value

A named list of default values

Methods (by class)

- `pgVirtual`: Default values for pgVirtual subclass objects
- `pgVirtual`: Set defaults for pgVirtual subclass objects

Examples

```
# Get all object defaults
testPG <- .loadPgExample()
defaults(testPG)

# Set a new default
defaults(testPG)$minFlank <- 2
```

`geneLocation` *Get gene location for all genes*

Description

This method returns the gene location of all genes as a data.frame with each row corresponding to a gene in the pangenome. The data.frame will have the columns 'start', 'end', 'contig' and 'strand' (order of columns not ensured) with start and end giving the start and end position of the gene on the contig/chromosome given in the contig column. Strand gives the direction of translation, 1 is from start to end and -1 is from end to start (thus start should always be lower than end no matter the direction of translation)

Usage

```
geneLocation(object)

## S4 method for signature 'pgInMemLoc'
geneLocation(object)
```

Arguments

`object`	A pgVirtual subclass

Value

A data.frame as described above

Methods (by class)

- `pgInMemLoc`: Get gene location for pgInMemLoc subclasses

Note

Required for subclasses of pgVirtualLoc in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE)
head(geneLocation(testPG))
```

geneNames *Get and set the names of the genes in the pangenome*

Description

These methods lets you query and change the naming of genes in your pangenome. Take note that even though sequences are not in memory for pgLM objects, the names are. This means that changes to the description header in the underlying fasta files have no effect on the naming in your pangenome

Usage

```
geneNames(object)

geneNames(object) <- value

## S4 method for signature 'pgLM'
geneNames(object)

## S4 replacement method for signature 'pgLM'
geneNames(object) <- value

## S4 method for signature 'pgFull'
geneNames(object)

## S4 replacement method for signature 'pgFull'
geneNames(object) <- value

## S4 method for signature 'pgSlim'
geneNames(object)

## S4 replacement method for signature 'pgSlim'
geneNames(object) <- value
```

Arguments

object	A pgVirtual subclass
value	A character vector with new names

Value

In case of the getter, a character vector containing the names of each gene.

Methods (by class)

- pgLM: Get genenames for pgLM and subclasses
- pgLM: Set genenames for pgLM and subclasses
- pgFull: Get genenames for pgFull and subclasses
- pgFull: Set genenames for pgFull and subclasses
- pgSlim: Throws error for pgSlim
- pgSlim: Throws error for pgSlim

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample()
head(geneNames(testPG))

geneNames(testPG)[10] <- 'Gene number 10'
```

---

genes *Extract gene sequences from a pangenome*

Description

This method is used to extract the genomic sequences that is the basis for the pangenome. Genes can be split and subsetted upfront based on other information in the pangenome, such as gene groups and organisms. For some pgVirtual subclasses the subset parameter is mandatory in order to avoid reading all genes into memory at once.

Usage

```
genes(object, split, subset)

## S4 method for signature 'pgLM,missing'
genes(object, split, subset)

## S4 method for signature 'pgLM,character'
genes(object, split, subset)

## S4 method for signature 'pgFull,missing'
genes(object, split, subset)

## S4 method for signature 'pgFull,character'
genes(object, split, subset)

## S4 method for signature 'pgSlim,missing'
genes(object, split, subset)

## S4 method for signature 'pgSlim,character'
genes(object, split, subset)
```

Arguments

object	A pgVirtual subclass
split	A string giving the optional splitting type. Either 'organism', 'group' or 'paralogue'.
subset	A subsetting of the result equal to using '[]' on the result. It is generally recommended to use this instead of subsetting the result, as it avoids unneeded memory allocation.

Value

An XStringSet if split is missing or an XStringSetList if it is present

Methods (by class)

- `object = pgLM,split = missing`: Gene access for pgLM and subclasses
- `object = pgLM,split = character`: Gene access for pgLM and subclasses with group splitting
- `object = pgFull,split = missing`: Gene access for pgFull and subclasses
- `object = pgFull,split = character`: Gene access for pgFull and subclasses with group splitting
- `object = pgSlim,split = missing`: Throws error for pgSlim
- `object = pgSlim,split = character`: Throws error for pgSlim

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample(withGroups=TRUE, withParalogues=TRUE)
# Direct gene access
genes(testPG)

# Early subsetting
genes(testPG, subset=1:10)

# Split by membership
genes(testPG, split='organism')
genes(testPG, split='group')
genes(testPG, split='paralogue')

# Split and subset - get genes from the first organism
genes(testPG, split='organism', subset=1)
```

---

geneWidth *Get the sequence length of each gene*

Description

This method extracts the width (i.e. number of residues) of each gene in the pangenome.

Usage

```
geneWidth(object)

## S4 method for signature 'pgLM'
geneWidth(object)

## S4 method for signature 'pgFull'
```

```
geneWidth(object)

## S4 method for signature 'pgSlim'
geneWidth(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer vector with the length of each sequence

Methods (by class)

- pgLM: Get gene width for pgLM and subclasses
- pgFull: Get gene widths for pgFull and subclasses
- pgSlim: Throws error for pgSlim

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample()
head(geneWidth(testPG))
```

---

getNeighborhood *Extract a graph representation of a gene group neighborhood*

Description

This method creates a graph representation of the imidiate neighborhood of a gene group. It is different from creating a subgraph of the panchromosome in that only vertices and edges directly reachable from the gene group is included. The vertices will be annotated with a centerGroup property indicating whether or not the node is the queried gene group.

Usage

```
getNeighborhood(object, ...)

## S4 method for signature 'pgVirtualLoc'
getNeighborhood(object, group, vicinity = 4)
```

Arguments

object	A pgVirtualLoc subclass
...	Parameters passed on.
group	Either the name or the index of the group whose neighborhood is of interest
vicinity	An integer giving the number of gene groups in both directions to collect

Value

An igraph object with gene groups as vertices and positional connections as edges. The edges is weighted according to the number of genes sharing the connection. All vertices have a centerGroup attribute, which is FALSE for all but the center group.

Methods (by class)

- `pgVirtualLoc`: Gene group neighborhoods for all pgVirtualLoc subclasses

See Also

`plotNeighborhood` for nice plotting of the neighborhood

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE, withNeighborhoodSplit=TRUE)

# Look at the surroundings of group 10
neighborhood <- getNeighborhood(testPG, group=10)
```

---

getRep *Get a representative sequence for each gene group*

Description

This method returns a representative sequence for each of the gene groups defined in the pangenome. Currently the methods defined for selecting sequences are 'random', 'shortest', and 'longest. In case of tie for the two latter the first occurence gets returned. Consensus sequence might be added at a latter stage.

Usage

```
getRep(object, method)

## S4 method for signature 'pgVirtual,character'
getRep(object, method)
```

Arguments

`object`	A pgVirtual subclass
`method`	The method to use to get a representative. Either 'random', 'shortest' or 'longest'.

Value

An XStringSet

Methods (by class)

- `object = pgVirtual,method = character`: Get a representative sequence for each gene group for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Get a random sequence from each group
getRep(testPG, 'random')
```

---

gpcGrouping *Guided Pairwise Comparison grouping of genes*

Description

This algorithm recursively builds up a pangenome by merging subpangenomes. The recursion follows either a supplied hierarchical clustering or one created using kmer comparison for the full organism. At each step a representative for each gene group is selected randomly as a representative and gets compared to all other representatives. Gene groups are then merged based on the pangenome created for the representatives. Due to the sampling of representatives at each step there is a certain randomness to the algorithm. Results should be fairly stable though, as gene groups are compared multiple times.

Usage

```
gpcGrouping(object, ...)

## S4 method for signature 'pgVirtual'
gpcGrouping(object, lowMem, kmerSize, tree, lowerLimit,
  pParam, cacheDB, precluster = TRUE, ...)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on.
lowMem	logical. Should low memory footprint be ensured over computation speed
kmerSize	The size of the kmer's used for the comparison. If two values are given and the 'tree' argument is missing, the second value is used for tree generation. If only one value is given it is recycled.
tree	An optional tree of class dendrogram (or that can be coerced to one) to guide the recursive algorithm. If none is supplied it will be generated by clustering the organisms by their total kmer numbers (summing up for each of their genes).
lowerLimit	A numeric giving the lower bounds of similarity below which it will be set to zero.
pParam	An optional BiocParallelParam object that defines the workers used for parallelisation.
cacheDB	A filehash object or a path to a directory where cached results should be stored. If omitted caching will not be done. Highly recommended for long running instances.
precluster	Logical. Should genes be preclustered using CD-Hit. Defaults to TRUE.

Value

An object of the same class as 'object'.

Methods (by class)

- `pgVirtual`: gpc grouping for all pgVirtual subclasses

See Also

Other grouping algorithms: `cdhitGrouping`, `graphGrouping`, `manualGrouping`

Examples

```
testPG <- .loadPgExample()

# Too heavy to include
## Not run:
testPG <- gpcGrouping(testPG)

## End(Not run)
```

---

graphGrouping *Use igraph to create gene grouping from a similarity matrix*

Description

This method takes a similarity matrix based on all genes in the pangenome, converts it to a graph representation and uses one of igraphs community detection algorithms to split all genes into groups. Within the FindMyFriends framework the similarity matrix would usually come from `kmerSimilarity`, but it can just as well be defined in other ways e.g. be blast derived.

Usage

```
graphGrouping(object, ...)

## S4 method for signature 'pgVirtual'
graphGrouping(object, similarity, algorithm, ...)
```

Arguments

`object`	A pgVirtual subclass
`...`	parameters to be passed on to the community detection algorithm
`similarity`	A similarity matrix with rows and columns corresponding to the genes in the pangenome.
`algorithm`	A string naming the algorithm. See `communities` for an overview. The trailing '.community' can be omitted from the name. Default is 'infomap', which is also the recommended.

Value

An object of the same class as 'object'.

Methods (by class)

- pgVirtual: graph grouping for all pgVirtual subclasses

See Also

Other grouping algorithms: cdhitGrouping, gpcGrouping, manualGrouping

Examples

```
testPG <- .loadPgExample()

# Too heavy to include
## Not run:
# Generate similarity matrix
simMat <- kmerSimilarity(testPG, lowerLimit=0.75)

# Group genes
testPG <- graphGrouping(testPG, simMat)

## End(Not run)
```

---

groupInfo *Get and set information about gene group*

Description

These methods lets you access the information stored about each gene group and add to it or modify it. Upfront the following columns are present: 'description', 'group', 'paralogue', 'GO', 'EC', 'nOrg' and 'nGenes'. All except 'group', 'nOrg' and 'nGenes' are filled with NA as default. The latter are prefilled with information derived from the grouping itself and should not be modified manually. 'description' is meant to contain a human readable description of the functionality of the gene group, 'GO' should contain GO terms (stored in a list of character vectors) and EC should contain enzyme numbers (again stored as a list of character vectors). There is no check for the validity of the content so it is up to the user to ensure that the terms added are valid. Additional columns can be added at will.

Usage

```
groupInfo(object)

groupInfo(object) <- value

## S4 method for signature 'pgInMem'
groupInfo(object)

## S4 replacement method for signature 'pgInMem'
groupInfo(object) <- value
```

Arguments

object	A pgVirtual subclass
value	A data.frame with a row for each group

Value

In case of the getter a data.frame with organism information.

Methods (by class)

- pgInMem: Get gene group metadata for pgInMem subclasses
- pgInMem: Set gene group metadata for pgInMem subclasses

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

See Also

Other Metadata: addGroupInfo, addOrgInfo, orgInfo

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)
head(groupInfo(testPG))

groupInfo(testPG)$description[1] <- 'transposase'
```

---

groupNames *Get and set the names of gene groups in the pangenome*

Description

These methods lets you manipulate the naming of gene groups in the pangenome. By default organisms are numbered consecutively but this can be changed at will. New gene groups will be numbered though despite what naming scheme has been introduced before.

Usage

```
groupNames(object)

groupNames(object) <- value

## S4 method for signature 'pgInMem'
groupNames(object)

## S4 replacement method for signature 'pgInMem'
groupNames(object) <- value
```

Arguments

object	A pgVirtual subclass
value	A vector with new names - will be coerced to characters

Value

In case of the getter a character vector with names

Methods (by class)

- pgInMem: Get gene group names for pgInMem subclasses
- pgInMem: Set gene group names for pgInMem subclasses

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)
head(groupNames(testPG))

groupNames(testPG)[20] <- 'Gene group 20'
```

---

groupStat *Calculate statistics about each gene group*

Description

This method calculates a range of statistics and positional information about each gene group. The information returned are. Maximum number of genes from the same organism (paralogues), shortest sequence length, longest sequence length, standard deviation of sequence lengths, index of genes in group, downstream and upstream gene groups.

Usage

```
groupStat(object, ...)

## S4 method for signature 'pgVirtual'
groupStat(object, vicinity = 1)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on.
vicinity	An integer given the number of flanking gene groups to traverse

Value

A list with an element for each gene group, each with the following elements.

maxOrg The highest number of distinct genes from the same organism present in the group. A number above 1 indicate the presence of paralogues.

minLength The length of the shortest sequence in the group.

maxLength The length of the longest sequence in the group.

sdLength The standard deviation of lengths in the group.

genes The index for the genes present in the group.

backward A character vector with gene groups separated by ';' that lies downstream of the gene group. The number of gene groups for each gene is controlled by the flankSize argument. If the contig stops before the required number of flanking genes have been reached, NA will be added. Downstream is defined in relation to the strand of the contig/chromosome, and not the translational direction of the gene in question.

forward As above in the other direction.

Methods (by class)

- `pgVirtual`: Group statistics for all pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

grStats <- groupStat(testPG)
```

---

hasGeneGroups *Check whether gene groups are defined*

Description

This method checks whether any grouping of genes has been done on the object and returns TRUE if that is the case.

Usage

```
hasGeneGroups(object)

## S4 method for signature 'pgVirtual'
hasGeneGroups(object)
```

Arguments

`object`	A pgVirtual subclass

Value

A boolean indicating whether gene groups have been defined (TRUE) or not (FALSE)

Methods (by class)

- `pgVirtual`: Gene group check for pgVirtual subclasses

Examples

```
# Empty pangenome
testPG <- .loadPgExample()
hasGeneGroups(testPG)

# With gene groups
testPG <- .loadPgExample(withGroups=TRUE)
hasGeneGroups(testPG)
```

`hasGeneInfo` *Checks for existance of gene location information*

Description

This method checks whether gene location information is present in the object i.e. if the object inherits from pgVirtualLoc

Usage

```
hasGeneInfo(object)

## S4 method for signature 'pgVirtual'
hasGeneInfo(object)
```

Arguments

`object`	A pgVirtual subclass

Value

A boolean indicating whether gene location information is present (TRUE) or not (FALSE)

Methods (by class)

- `pgVirtual`: Checks whether gene location information is available for pgVirtual subclasses

Examples

```
# Exclusive pgVirtual subclasses
testPG <- .loadPgExample()
hasGeneInfo(testPG)

# pgVirtualLoc subclasses
testPG <- .loadPgExample(geneLoc=TRUE)
hasGeneInfo(testPG)
```

hasParalogueLinks *Checks whether linking of paralogues has been done*

Description

This method checks for the existance of paralogue links in the object.

Usage

```
hasParalogueLinks(object)

## S4 method for signature 'pgVirtual'
hasParalogueLinks(object)
```

Arguments

object	A pgVirtual subclass

Value

A boolean indicating whether paralogue links have been defined (TRUE) or not (FALSE)

Methods (by class)

- pgVirtual: Check for secondary gene grouping in pgVirtual subclasses

Examples

```
# No paralogues
testPG <- .loadPgExample(withGroups=TRUE)
hasParalogueLinks(testPG)

# With paralogues
testPG <- .loadPgExample(withGroups=TRUE, withParalogues=TRUE)
hasParalogueLinks(testPG)
```

kmerLink *Link gene groups by homology*

Description

This method allows the user to define a secondary grouping of genes be linking gene groups based on sequence similarity (paralogues). A representative for each gene group is used for the calculations and the similarity is assessed using the kmer based cosine similarity.

Usage

```
kmerLink(object, ...)

## S4 method for signature 'pgVirtual'
kmerLink(object, lowMem, kmerSize, lowerLimit, rescale,
  transform, pParam, algorithm, ...)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on to the community detection algorithm.
lowMem	logical. Should low memory footprint be ensured over computation speed
kmerSize	The size of kmers to use for similarity calculations.
lowerLimit	The lower threshold for similarity below which it is set to 0
rescale	Should Similarities be normalised between lowerLimit and 1
transform	Transformation function to apply to similarities
pParam	An optional BiocParallelParam object that defines the workers used for parallelisation.
algorithm	The name of the community detection algorithm from igraph to use for gene grouping. See communities for an overview. The trailing '.community' can be omitted from the name. Default is 'infomap', which is also the recommended.

Value

An object with the same class as object with linking between gene groups.

Methods (by class)

- pgVirtual: Linking for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# No paralogue links
hasParalogueLinks(testPG)

# Create the links
testPG <- kmerLink(testPG)
```

---

kmerSimilarity *Calculate a similarity matrix based on kmers*

Description

This method takes a pangenome and calculate a similarity matrix based on cosine similarity of kmer feature vectors in an all-vs-all fashion. The result can subsequently be used to group genes either using graphGrouping or homebrewed grouping scheme. In case of the latter manualGrouping should be used to add the grouping back to the pangenome.

Usage

```
kmerSimilarity(object, ...)

## S4 method for signature 'pgVirtual'
kmerSimilarity(object, lowMem, kmerSize, lowerLimit,
  rescale, transform, pParam)
```

Arguments

object	A pgVirtual subclass
...	parameters passed on.
lowMem	logical. Should low memory footprint be ensured over computation speed
kmerSize	The size of kmers to use for similarity calculations.
lowerLimit	The lower threshold for similarity below which it is set to 0
rescale	Should Similarities be normalised between lowerLimit and 1
transform	Transformation function to apply to similarities
pParam	An optional BiocParallelParam object that defines the workers used for parallelisation.

Value

A matrix (sparse or normal) with cosine similarity for each gene pair

Methods (by class)

- pgVirtual: Kmer based similarities for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample()

# Too heavy to include
## Not run:
kmerSim <- kmerSimilarity(testPG, lowerLimit=0.75)

## End(Not run)
```

---

kmerSplit *Split gene groups based on similarity*

Description

This function splits up gene groups based on cosine similarity of kmer feature vectors. It uses hard splitting based on a similarity cutoff where unconnected components constitutes new groups. Unlike neighborhoodSplit, paralogues cannot be forced into separate groups as information needed for this is not present.

Usage

```
kmerSplit(object, ...)

## S4 method for signature 'pgVirtual'
kmerSplit(object, kmerSize, lowerLimit, maxLengthDif,
  pParam)
```

Arguments

object	A pgVirtual subclass
...	Arguments passed on
kmerSize	The length of kmers used for sequence similarity
lowerLimit	The lower limit of sequence similarity below which it will be set to 0
maxLengthDif	The maximum deviation in sequence length to allow. Between 0 and 1 it describes a percentage. Above 1 it describes a fixed length
pParam	An optional BiocParallelParam object that defines the workers used for parallelisation.

Value

A new pgVirtual subclass object of the same class as 'object'

Methods (by class)

- pgVirtual: Kmer similarity based group splitting for pgVirtual subclasses

See Also

Other group-splitting: neighborhoodSplit

Examples

```
# Get a grouped pangenome
pg <- .loadPgExample(withGroups = TRUE)

## Not run:
# Split groups by similarity (Too heavy to include)
pg <- kmerSplit(pg, lowerLimit = 0.8)

## End(Not run)
```

---

manualGrouping *Define gene grouping manually*

Description

In cases where results from other algorithms are wished to be imported into the FindMyFriends framework, this method ensures that the proper formatting is done. The grouping can be defined as an integer vector with an element for each gene. The value of each element is then used as the gene group classifier. Alternatively groups can be defined by a list of integer vectors. Each element of the list defines a group and the content of each element refers to gene indexes.

Usage

```
manualGrouping(object, groups)

## S4 method for signature 'pgVirtual,integer'
manualGrouping(object, groups)

## S4 method for signature 'pgVirtual,list'
manualGrouping(object, groups)
```

Arguments

object	A pgVirtual subclass
groups	Either a list or integer vector defining the grouping

Value

An object of the same class as 'object'.

Methods (by class)

- `object = pgVirtual,groups = integer`: manual grouping defined by integer vector
- `object = pgVirtual,groups = list`: manual grouping defined by list

See Also

Other grouping algorithms: `cdhitGrouping`, `gpcGrouping`, `graphGrouping`

Examples

```
testPG <- .loadPgExample()

# Load grouping data
groups <- system.file('extdata', 'examplePG', 'groupsWG.txt',
    package='FindMyFriends'
)
groups <- scan(groups, what=integer(), quiet=TRUE)

# Do the grouping
testPG <- manualGrouping(testPG, groups)
```

---

neighborhoodSplit *Split gene groups by neighborhood synteny*

Description

This function evaluates already created gene groups and splits the members into new groups based on the synteny of the flanking genes and the similarity of the sequences. In general the splitting is based on multiple stages that all gene pairs must pass in order to remain in the same group. First the link between the genes is removed if they are part of the same organism. Then the synteny of the flanking genes are assessed and if it doesn't passes the defined threshold the link between the gene pair is removed. Then the kmer similarity of the two sequences are compared and if below

a certain threshold the link is removed. Lastly the length of the two sequences are compared and if below a certain threshold the link is removed. Based on this new graph cliques are detected and sorted based on the lowest within-clique sequence similarity and neighborhood synteny. The cliques are then added as new groups if the members are not already members of a new group until all members are part of a new group. This approach ensures that all members of the new groupings passes certain conditions when compared to all other members of the same group. After the splitting a refinement step is done where gene groups with high similarity and sharing a neighbor either up- or downstream are merged together to avoid spurius errors resulting from the initial grouping.

Usage

```
neighborhoodSplit(object, ...)

## S4 method for signature 'pgVirtualLoc'
neighborhoodSplit(object, flankSize, forceParalogues,
  kmerSize, lowerLimit, maxLengthDif, guideGroups = NULL,
  cdhitOpts = list())
```

Arguments

`object`	A pgVirtualLoc subclass
`...`	parameters passed on.
`flankSize`	The number of flanking genes on each side of the gene to use for comparison.
`forceParalogues`	Force similarity of paralogue genes to 0
`kmerSize`	The length of kmers used for sequence similarity
`lowerLimit`	The lower limit of sequence similarity below which it will be set to 0
`maxLengthDif`	The maximum deviation in sequence length to allow. Between 0 and 1 it describes a percentage. Above 1 it describes a fixed length
`guideGroups`	An integer vector with prior grouping that, all else being equal, should be prioritized. Used internally.
`cdhitOpts`	A list of options to pass on to CD-Hit during the merging step. "l", "n" and "s"/"S" will be overridden.

Value

An object with the same class as object containing the new grouping.

Methods (by class)

- `pgVirtualLoc`: Neighborhood-based gene group splitting for pgVirtualLoc subclasses

See Also

Other group-splitting: `kmerSplit`

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE, withGroups=TRUE)

# Too heavy to run
## Not run:
testPG <- neighborhoodSplit(testPG, lowerLimit=0.75)

## End(Not run)
```

---

nGeneGroups *Get the number of gene groups in a pangenome*

Description

This method gives the number of different gene groups in the object.

Usage

```
nGeneGroups(object)

## S4 method for signature 'pgVirtual'
nGeneGroups(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer giving the number of gene groups

Methods (by class)

- pgVirtual: The number of gene groups in the pangenome for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)
nGeneGroups(testPG)
```

nGenes *Get the total number of genes in a pangenome*

Description

This method returns the total number of genes in a pangenome (i.e. the sum of genes in each organism in the pangenome)

Usage

```
nGenes(object)

## S4 method for signature 'pgVirtual'
nGenes(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer giving the number of genes in the object

Methods (by class)

- pgVirtual: The number of genes in the pangenome for pgVirtual subclasses.

Examples

```
testPG <- .loadPgExample()
nGenes(testPG)
```

nOrganisms *Get the number of organisms represented in a pangenome*

Description

This method returns the current number of organisms in a pgVirtual subclass. This is also the result of calling `length()` on the object.

Usage

```
nOrganisms(object)

## S4 method for signature 'pgVirtual'
nOrganisms(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer giving the number of organisms

Methods (by class)

- `pgVirtual`: The number of organisms in the pangenome for pgVirtual subclasses.

Examples

```
testPG <- .loadPgExample()
nOrganisms(testPG)
```

`orgInfo` *Get and set information about organisms*

Description

These methods lets you access the information stored about each organism and add to it or modify it. The only information present up front is the number of genes present in each organism. While possible, this information should not be changed manually but through the `removeGene` functions.

Usage

```
orgInfo(object)

orgInfo(object) <- value

## S4 method for signature 'pgInMem'
orgInfo(object)

## S4 replacement method for signature 'pgInMem'
orgInfo(object) <- value
```

Arguments

`object`	A pgVirtual subclass
`value`	A data.frame with a row for each organism

Value

In case of the getter a data.frame with organism information.

Methods (by class)

- `pgInMem`: Get organism metadata for pgInMem subclasses
- `pgInMem`: Set organism metadata for pgInMem subclasses

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

See Also

Other Metadata: addGroupInfo, addOrgInfo, groupInfo

Examples

```
testPG <- .loadPgExample()
orgInfo(testPG)

orgInfo(testPG)$Genus <- 'Mycoplasma'
```

---

orgNames *Get and set the names of organisms in the pangenome*

Description

These methods lets you manipulate the naming of organisms in the pangenome. By default organisms are named after the fasta file they are defined by, but this can be changed at will.

Usage

```
orgNames(object)

orgNames(object) <- value

## S4 method for signature 'pgInMem'
orgNames(object)

## S4 replacement method for signature 'pgInMem'
orgNames(object) <- value
```

Arguments

object	A pgVirtual subclass
value	A vector with new names - will be coerced to characters

Value

In case of the getter a character vector with names

Methods (by class)

- pgInMem: Get organism names for pgInMem subclasses
- pgInMem: Set organism names for pgInMem subclasses

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample()
orgNames(testPG)

orgNames(testPG)[3] <- 'Organism 3'
```

`orgStat` *Calculate statistics about each organism*

Description

This method, much like codegroupStat calculates different statistics for each organism in the pangenome. Depending on the parameters the statistics are: number of genes, minimum length of gene, maximum length of gene standard deviation of gene lengths, residue frequency, number of gene groups and number of paralogues.

Usage

```
orgStat(object, ...)

## S4 method for signature 'pgVirtual'
orgStat(object, subset, getFrequency = FALSE)
```

Arguments

`object`	A pgVirtual subclass
`...`	parameters passed on.
`subset`	Name or indexes of organisms to include
`getFrequency`	logical. Should amino/nucleic acid frequency be calculated

Value

A data.frame with a row per organism, with each statistic in a column

Methods (by class)

- `pgVirtual`: Organism statistics for all pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

orgStats <- orgStat(testPG)
```

pangenome *Construct a pangenome from fasta files*

Description

This function constructs an initial pangenome object from a set of fasta files. Note that the actual pangenome is not calculated here. As such this function mainly sets everything up before beginning the more lengthly pangenome calculation.

Usage

```
pangenome(paths, translated, geneLocation = NULL, lowMem = FALSE, ...)
```

Arguments

paths	A character vector with location of fasta files
translated	A boolean indicating if the fasta files contain amino acid sequences
geneLocation	A function, string or dataframe. If it is a data.frame it should contain the columns 'contig', 'start', 'end' and 'strand' with a row for each gene. If it is a function it should take the name (fasta description) for each gene and output a data.frame similar to described above. If it is a string it should specify the format of the gene names. Currently only 'prodigal' is supported.
lowMem	Boolean. Should FindMyFriends avoid storing sequences in memory.
...	Additional defaults to set on the object

Value

A pgVirtual subclass object depending on geneLocation and lowMem.

geneLocation	lowMem	Resulting class
NULL	FALSE	pgFull
NULL	TRUE	pgLM
!NULL	FALSE	pgFullLoc
!NULL	TRUE	pgLMLoc

Examples

```
location <- tempdir()
unzip(system.file('extdata', 'Mycoplasma.zip', package='FindMyFriends'),
      exdir=location)
genomeFiles <- list.files(location, full.names=TRUE, pattern='*.fasta')

# Create pgFull
pangenome(genomeFiles, TRUE)

# Create pgFullLoc
pangenome(genomeFiles, TRUE, geneLocation='prodigal')

# Create pgLM
pangenome(genomeFiles, TRUE, lowMem=TRUE)
```

```
# Create pgLMLoc
pangenome(genomeFiles, TRUE, geneLocation='prodigal', lowMem=TRUE)
```

---

pcGraph *Calculate the panchromosome graph*

Description

This method creates a graph representation of the panchromosome - The complete set of gene groups linked together by chromosomal position. Each vertice in the graph represent a gene group and each edge represent a positional relation between two gene groups (neighboring each other). Vertices are annotated with number of genes, organism names and strand while edges are annotated with numer of genes (as weight), and organism names.

Usage

```
pcGraph(object, ...)

## S4 method for signature 'pgVirtualLoc'
pcGraph(object, slim = FALSE)
```

Arguments

object	A pgVirtualLoc subclass
...	parameters passed on
slim	Should the returned graph be stripped of all metadata and only capture gene group connectivity. Defaults to FALSE

Value

An igraph object

Methods (by class)

- pgVirtualLoc: Panchromosome creation for all pgVirtualLoc subclasses

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE, withNeighborhoodSplit=TRUE)

panchromosome <- pcGraph(testPG)
```

pgFull-class *Class for in memory pangenome data*

Description

This class handles pangenome data without gene location information and with all sequences stored in memory. This makes sequence lookup much faster but also increases the memory footprint of the object thus making it a bad choice for very large pangenome with millions of genes.

Slots

`sequences` Either an AAStringSet or DNAStringSet containing all sequences in the pangenome.

See Also

Other Pangenome_classes: `pgFullLoc-class`, `pgInMem-class`, `pgInMemLoc-class`, `pgLM-class`, `pgLMLoc-class`, `pgSlim-class`, `pgSlimLoc-class`, `pgVirtual-class`, `pgVirtualLoc-class`

pgFullLoc-class *Class for in memory pangenome data with location information*

Description

This class extends `pgFull` by subclassing `pgInMemLoc` and thus adding gene location information to each gene. See the respective superclasses for more information.

See Also

Other Pangenome_classes: `pgFull-class`, `pgInMem-class`, `pgInMemLoc-class`, `pgLM-class`, `pgLMLoc-class`, `pgSlim-class`, `pgSlimLoc-class`, `pgVirtual-class`, `pgVirtualLoc-class`

pgInMem-class *FindMyFriends standard base class for pangenomic data*

Description

This virtual class is the superclass of the standard pangenome classes in FindMyFriends. It defines storage for everything except gene information, which is delegated to its subclasses.

Details

As gene storage is not defined in this class the following methods must be defined by subclasses:

genes(object, split, subset) Return the underlying sequences. If split is missing return an XStringSet, otherwise return an XStringSetList. split can be either 'group', 'organism' or 'paralogue' and should group the sequences accordingly. Subset should behave as if it was added as '[]' to the results but allow you to avoid reading everything into memory if not needed.

geneNames(object) Return a character vector with the name of each gene.

geneNames<-(object, value) Set the name of each gene.

geneWidth(object) Return an integer vector with the length (in residues) of each gene.

removeGene(object, name, organism, group, ind) Should only be implemented for signature: c(*yourClass*, 'missing', 'missing', 'missing', 'integer') Remove the genes at the given indexes and return the object.

Slots

`seqToOrg` An integer vector that reference all genes to a specific organism.

`seqToGeneGroup` An integer vector that references all genes to a specific gene group.

`groupInfo` A data.frame storing metadata information about gene groups.

`orgInfo` A data.frame storing metadata information about organisms

See Also

Other Pangenome_classes: `pgFull-class`, `pgFullLoc-class`, `pgInMemLoc-class`, `pgLM-class`, `pgLMLoc-class`, `pgSlim-class`, `pgSlimLoc-class`, `pgVirtual-class`, `pgVirtualLoc-class`

`pgInMemLoc-class` *Superclass for gene location aware pangenome*

Description

This virtual class is the superclass for all standard, location aware, pangenome classes in FindMyFriends. It stores all chromosomal information in a data.frame.

Slots

`geneLocation` A data.frame containing the columns 'contig', 'start', 'end' and 'strand' and a row for each gene in the pangenome.

See Also

Other Pangenome_classes: `pgFull-class`, `pgFullLoc-class`, `pgInMem-class`, `pgLM-class`, `pgLMLoc-class`, `pgSlim-class`, `pgSlimLoc-class`, `pgVirtual-class`, `pgVirtualLoc-class`

pgLM-class *Class for reference based pangenome data*

Description

This class handles pangenome information where gene sequences are kept on disc instead of stored in memory. As long as the original fasta files are not modified, this class will take care of indexing the genes correctly. This class has a substantially lower memory footprint than the pgFull class at the expense of longer sequence lookup times. For massive pangenomes containing Gb of sequence data there is no alternative though.

Slots

seqIndex A data.frame as produced by fasta.index with random access information for each gene.

See Also

Other Pangenome_classes: pgFull-class, pgFullLoc-class, pgInMem-class, pgInMemLoc-class, pgLMLoc-class, pgSlim-class, pgSlimLoc-class, pgVirtual-class, pgVirtualLoc-class

pgLMLoc-class *Class for reference based pangenome data with location information*

Description

This class extends pgLM by subclassing pgInMemLoc and thus adding gene location information to each gene. See the respective superclasses for more information.

See Also

Other Pangenome_classes: pgFull-class, pgFullLoc-class, pgInMem-class, pgInMemLoc-class, pgLM-class, pgSlim-class, pgSlimLoc-class, pgVirtual-class, pgVirtualLoc-class

pgMatrix *Get the pangenome matrix*

Description

This method lets you extract the pangenome matrix of the pangenome. It is not possible to directly change the pangenome matrix as it not necessary stored in the object but might be calculated on request. Either way the pangenome matrix is a function of the gene grouping and should be changed by changing the gene grouping instead of being manipulated downstream.

Usage

```
pgMatrix(object)

## S4 method for signature 'pgVirtual'
pgMatrix(object)
```

Arguments

object	A pgVirtual subclass

Value

A matrix with organisms as columns and gene groups as rows

Methods (by class)

- pgVirtual: Get pangenome matrix for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

head(pgMatrix(testPG))
```

---

pgSlim-class *Class for pangenome data with no reference to genes*

Description

This class is a slim version of pgLM and pgFull that does not store any information pertaining to the actual genes. This means that this class cannot be the basis for the creation of a pangenome but that pgLM or pgFull objects can be coerced down to this representation after the pangenome has been created to make it less burdensome to work with, while still keeping a lot of the functionality of the FindMyFriends framework.

See Also

Other Pangenome_classes: pgFull-class, pgFullLoc-class, pgInMem-class, pgInMemLoc-class, pgLM-class, pgLMLoc-class, pgSlimLoc-class, pgVirtual-class, pgVirtualLoc-class

pgSlimLoc-class *Class for pangenome data with no reference to genes*

Description

This class extends pgSlim by subclassing pgInMemLoc and thus adding gene location information to each gene. See the respective superclasses for more information.

See Also

Other Pangenome_classes: pgFull-class, pgFullLoc-class, pgInMem-class, pgInMemLoc-class, pgLM-class, pgLMLoc-class, pgSlim-class, pgVirtual-class, pgVirtualLoc-class

pgVirtual-class *Base class for pangenomic data*

Description

This virtual class is the superclass of all other pangenome classes in FindMyFriends. It is an empty shell that is mainly used for dispatch and checking that the promises of subclasses are held.

Usage

```
## S4 method for signature 'pgVirtual'
length(x)

## S4 method for signature 'pgVirtual'
show(object)

## S4 method for signature 'pgVirtual,integer,ANY,ANY'
x[i]

## S4 method for signature 'pgVirtual,numeric,ANY,ANY'
x[i]

## S4 method for signature 'pgVirtual,character,ANY,ANY'
x[i]

## S4 method for signature 'pgVirtual,logical,ANY,ANY'
x[i]

## S4 method for signature 'pgVirtual,ANY,ANY'
x[[i]]

as(object, Class='ExpressionSet')

as(object, Class='matrix')
```

Arguments

x	A pgVirtual subclass object
object	A pgVirtual subclass object
i	indices specifying genomes, either integer, numeric, character or logical, following the normal rules for indexing objects in R
Class	The class to coerce pgVirtual subclasses to. Outside of the FindMyFriends class tree only 'ExpressionSet' and 'matrix' is implemented.

Details

Subclasses of pgVirtual must implement the following methods in order for them to plug into FindMyFriends algorithms:

seqToOrg(object) Returns the mapping from genes to organisms as an integer vector with position mapped to gene and integer mapped to organism.

seqToGeneGroup(object) As seqToOrg but mapped to gene group instead of organism. If gene groups are yet to be defined return an empty vector.

genes(object, split, subset) Return the underlying sequences. If split is missing return an XStringSet, otherwise return an XStringSetList. split can be either 'group', 'organism' or 'paralogue' and should group the sequences accordingly. Subset should behave as if it was added as '[]' to the results but allow you to avoid reading everything into memory if not needed.

geneNames(object) Return a character vector with the name of each gene.

geneNames<-(object, value) Set the name of each gene.

geneWidth(object) Return an integer vector with the length (in residues) of each gene.

removeGene(object, name, organism, group, ind) Should only be implemented for signature: c(*yourClass*, 'missing', 'missing', 'missing', 'integer') Remove the genes at the given indexes and return the object.

orgNames(object) Return a character vector of organism names.

orgNames<-(object, value) Set the name of the organisms.

groupNames(object) Return a character vector of gene group names.

groupNames<-(object, value) Set the name of gene groups.

orgInfo(object) Return a data.frame with metadata about each organim.

orgInfo<-(object, value) Set a data.frame to be metadata about each organism.

setOrgInfo(object, name, info, key) Set the metadata 'name', for the organisms corresponding to 'key' to 'info'

groupInfo(object) Return a data.frame with metadata about each gene group.

groupInfo<-(object, value) Set a data.frame to be metadata about each gene group.

setGroupInfo(object, name, info, key) Set the metadata 'name', for the gene groups corresponding to 'key' to 'info'

groupGenes(object, seqToGeneGroup) Sets the gene grouping of the pangenome. 'seqToGeneGroup' should correspond to the output of the seqToGeneGroup method (i.e. an integer vector with each element giving the group of the corresponding gene). This method **must** include a `callNextMethod(object)` as the last line.

mergePangenomes(pg1, pg2, geneGrouping, groupInfo) Merge pg2 into pg1 preserving the indexing in pg1 and appending and modifying the indexing of pg2. The geneGrouping argument is the new grouping of genes and groupInfo the new group info for the groups.

Additionally subclasses can override the following methods for performance gains. Otherwise they will be derived from the above methods.

length(object) Return the number of organisms in the object.

nOrganisms(object) As length.

nGenes(object) Return the number of genes in the object.

nGeneGroups(object) Return the number of gene groups

hasGeneGroups Returns TRUE if gene groups have been defined

pgMatrix Returns an integer matrix with organisms as columns and gene groups as rows and the corresponding number of genes in each element.

Developers are encourages to consult the implementation of FindMyFriends own classes when trying to implement new ones

Value

Length returns an integer giving the number of organisms

Methods (by generic)

- `length`: Length of a Pangenome, defined as the number of organisms it contain
- `show`: Basic information about the pangenome
- `[`: Create subsets of pangenomes based on index
- `[`: Create subsets of pangenomes based on index
- `[`: Create subsets of pangenomes based on organism name
- `[`: Create subsets of pangenomes based on logical vector
- `[[`: Extract sequences from a single organism

Slots

`.settings` A list containing settings pertaining to the object

See Also

Other Pangenome_classes: `pgFull-class`, `pgFullLoc-class`, `pgInMem-class`, `pgInMemLoc-class`, `pgLM-class`, `pgLMLoc-class`, `pgSlim-class`, `pgSlimLoc-class`, `pgVirtualLoc-class`

`pgVirtualLoc-class` *Superclass for gene location aware pangenome*

Description

This virtual class should be subclassed by all classes that include chromosomal position of the genes (along with subclassing pgVirtual). The class itself is an empty shell that only takes care of dispatching and checking the promises of subclasses are held.

Details

Subclasses of pgVirtualLoc must implement the following methods:

geneLocation(object) Return a data.frame with a row for each gene, describing the chromosomal position of the gene. The data.frame must contain the columns 'contig', 'start', 'end' and 'strand'. Contig is self-explanatory, start and end is the respective start and end positions on the contig (start must be lower than end) and strand defines the coding direction as -1 or 1.

See Also

Other Pangenome_classes: pgFull-class, pgFullLoc-class, pgInMem-class, pgInMemLoc-class, pgLM-class, pgLMLoc-class, pgSlim-class, pgSlimLoc-class, pgVirtual-class

---

plotEvolution *Plot the evolution in gene groups*

Description

This method constructs a plot showing how the number of singleton, accessory and core gene groups evolve as the size of the pangenome increases. Different ways of increasing the size of the pangenome is available.

Usage

```
plotEvolution(object, ...)

## S4 method for signature 'pgVirtual'
plotEvolution(object, ordering = "bootstrap",
  times = 10)
```

Arguments

object	A pgVirtual subclass
...	Parameters to be passed on
ordering	An ordering of the organisms by name or index, or alternative one of 'bootstrap', 'random' or 'none'.
times	The number of sampling for ordering='bootstrap'

Value

This function is called for its side effects

Methods (by class)

- pgVirtual: Evolution plots for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Standard type - organisms ordered by their index in the pangenome
plotEvolution(testPG, ordering='none')

# Bootstrapped with confidence intervals
plotEvolution(testPG, ordering='bootstrap')
```

`plotGroup` *Plot the similarities of genes within a group*

Description

This method plots a gene group with genes as vertices and cosine similarities as weighted edges. Mildly informative at best :-)

Usage

```
plotGroup(object, ...)

## S4 method for signature 'pgVirtual'
plotGroup(object, group, kmerSize, lowerLimit, rescale,
  transform, ...)
```

Arguments

`object`	A pgVirtual subclass
`...`	Parameters to be passed on to igraphs plotting method
`group`	Name or index of the gene group to plot
`kmerSize`	The kmer size to use for similarity calculations
`lowerLimit`	The lower threshold for similarity below which it will be set to 0
`rescale`	logical. Should the similarity be rescaled between lowerLimit and 1
`transform`	A transformation function to apply to the similarities

Value

Called for the side effect of creating a plot. Invisibly returns an igraph object with all visual parameters set as node and edge attributes.

Methods (by class)

- `pgVirtual`: Gene group similiarity plotting for all pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

plotGroup(testPG, 10, lowerLimit=0.25)
```

plotNeighborhood *Plot the neighborhood of a gene group*

Description

This method plots the neighborhood extracted using `getNeighborhood` in a visually pleasing way. It is mainly a wrapper around `plot.igraph` to ensure the proper information is visualised.

Usage

```
plotNeighborhood(object, ...)

## S4 method for signature 'pgVirtualLoc'
plotNeighborhood(object, group, vicinity = 4, ...)
```

Arguments

`object`	A pgVirtualLoc subclass
`...`	Parameter passed on to igraph's plot method.
`group`	The name or index of a group.
`vicinity`	An integer giving the number of gene groups in both directions to collect.

Value

Called for the side effect of creating a plot. Invisibly returns an igraph object with all visual parameters set as node and edge attributes.

Methods (by class)

- `pgVirtualLoc`: Gene group neighborhood plotting for all pgVirtualLoc subclasses

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE, withNeighborhoodSplit=TRUE)

# Nice little overview of the neighborhood of gene group 30
plotNeighborhood(testPG, 30)
```

plotSimilarity *Create a heatplot with similarities between all organisms*

Description

This method creates a heatplot showing the similarity between all organisms in the pangenome. The similarity can either be derived from the pangenome matrix or from kmer calculations of the genes themselves.

Usage

```
plotSimilarity(object, ...)

## S4 method for signature 'pgVirtual'
plotSimilarity(object, type = "pangenome",
  ordering = "auto", kmerSize, pParam, chunkSize = 100)
```

Arguments

object	A pgVirtual subclass
...	Parameters to be passed on.
type	The type of similarity calculation. Either 'pangenome' or 'kmer'
ordering	The ordering of rows and column in the heatmap. Either integer og character vector with organism names or one of the following: 'auto' or 'none'. For 'auto' The ordering will be based on a hierarchical clustering of the organisms based on Ward's distance.
kmerSize	The size of the kmers to use for comparison
pParam	An object of class BiocParallelParam
chunkSize	Number of organisms to process at a time

Value

This function is called for its side effects

Methods (by class)

- pgVirtual: Similarity heatmaps for pgVirtual subclasses

See Also

plotTree for a dendrogram plot of the same data.

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Use kmers
plotSimilarity(testPG, type='kmer')

# Use pangenome matrix
plotSimilarity(testPG, type='pangenome')
```

plotStat *Plot (very) basic statistics on the pangenome*

Description

This method plots the number of genes in each organism and, if gene groups have been defined, the number of singleton, accessory and core gene groups.

Usage

```
plotStat(object, ...)

## S4 method for signature 'pgVirtual'
plotStat(object, sort = TRUE, color, ...)
```

Arguments

object	A pgVirtual subclass
...	Parameters passed on to color scale.
sort	logical. Should Genomes be sorted based on their number of genes
color	A metadata name to color the organisms by

Value

This function is called for its side effects

Methods (by class)

- pgVirtual: Plot basic statistics for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Should make a nice little plot
plotStat(testPG)
```

plotTree *Plot a dendrogram of the organisms in a pangenome*

Description

This method plots a dendrogram of the relationship between the organisms in the pangenome. It does not tries to by phylogenetic in any way but merely shows the relationship in data. As with plotSimilarity it can be based on either the pangenome matrix or kmer feature vectors.

Usage

```
plotTree(object, ...)

## S4 method for signature 'pgVirtual'
plotTree(object, type = "pangenome", circular = FALSE,
  info, kmerSize, dist, clust, pParam, chunkSize = 100)
```

Arguments

object	A pgVirtual subclass
...	Parameters to be passed on.
type	The type of distance meassure. Either 'pangenome' or 'kmer'
circular	logical. Should the dendrogram be plotted in a circular fashion.
info	Metadata to plot at the leafs of the dendrogram. Either the name of an orgInfo column or a vector with information for each organism.
kmerSize	The size of the kmers to use for comparison
dist	The distance function to use. All possible values of method in dist() are allowed as well as 'cosine' for type='kmer'
clust	The clustering function to use. Passed on to method in hclust()
pParam	An object of class BiocParallelParam
chunkSize	Number of organisms to process at a time

Value

This function is called for its side effects

Methods (by class)

- pgVirtual: Dendrogram plotting for pgVirtual subclasses

See Also

plotSimilarity for a heatmap plot of the same data.

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

plotTree(testPG, type='pangenome', dist='binary', clust='ward.D2')

# And now in a circle (type defaults to 'pangenome')
plotTree(testPG, circular=TRUE, dist='binary', clust='ward.D2')
```

readAnnot *Import annotation from an .annot file*

Description

This function makes it easy to import annotation create in Blast2GO or other programs supporting .annot exporting of results.

Usage

```
readAnnot(file)
```

Arguments

file	The .annot file to import

Value

A data.frame ready to merge with a pangenome object using addGroupInfo with the key argument set to 'name'.

Examples

```
# Get path to file
annot <- system.file('extdata', 'examplePG', 'example.annot', package='FindMyFriends')

# Parse the file
readAnnot(annot)
```

removeGene *Remove genes from a pangenome*

Description

This method makes it possible to safely remove genes from a pangenome using a variaty of selection mechanisms depending on the supplied parameters. The name parameter refers to the gene name, organism refers to either organism name or index, group refers to either gene group name or index and ind refers to the gene index. See examples for details of the different possibilities.

Usage

```
removeGene(object, name, organism, group, ind, ...)

## S4 method for signature 'pgInMem,missing,missing,missing,numeric'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,character,missing,missing,missing'
removeGene(object, name,
```

```
  organism, group, ind)

## S4 method for signature 'pgVirtual,character,character,missing,missing'
removeGene(object,
  name, organism, group, ind)

## S4 method for signature 'pgVirtual,character,numeric,missing,missing'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,character,missing,missing'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,numeric,missing,missing'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,character,missing,numeric'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,numeric,missing,numeric'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,missing,character,missing'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,missing,numeric,missing'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,missing,character,numeric'
removeGene(object, name,
  organism, group, ind)

## S4 method for signature 'pgVirtual,missing,missing,numeric,numeric'
removeGene(object, name,
  organism, group, ind)
```

Arguments

object	A pgVirtual subclass
name	A character vector of names of genes to remove
organism	Either an integer or character vector of orgnanisms to remove genes from. If neither name nor ind is given all genes in the organisms are removed.
group	Either an integer or character vector of gene groups to remove genes from. If ind is not given all genes in the groups are removed.
ind	Indexes of the selections to remove. If both name, organism and group is not given, it indexes into the raw gene index, otherwise it indexes into the element

defined by organism or group.

... parameters passed on (currently ignored).

Value

An object of the same class as object without the genes that should be removed.

Methods (by class)

- `object = pgInMem,name = missing,organism = missing,group = missing,ind = numeric`: Gene removal base function for pgInMem subclasses
- `object = pgVirtual,name = character,organism = missing,group = missing,ind = missing`: Remove gene based on gene name
- `object = pgVirtual,name = character,organism = character,group = missing,ind = missing`: Remove gene based on gene and organism name
- `object = pgVirtual,name = character,organism = numeric,group = missing,ind = missing`: Remove gene based on gene name and organism index
- `object = pgVirtual,name = missing,organism = character,group = missing,ind = missing`: Remove gene based on organism name
- `object = pgVirtual,name = missing,organism = numeric,group = missing,ind = missing`: Remove gene based on organism index
- `object = pgVirtual,name = missing,organism = character,group = missing,ind = numeric`: Remove gene based on organism name and gene index
- `object = pgVirtual,name = missing,organism = numeric,group = missing,ind = numeric`: Remove gene based on organism and gene index
- `object = pgVirtual,name = missing,organism = missing,group = character,ind = missing`: Remove gene based on gene group name
- `object = pgVirtual,name = missing,organism = missing,group = numeric,ind = missing`: Remove gene based on gene group index
- `object = pgVirtual,name = missing,organism = missing,group = character,ind = numeric`: Remove gene based on gene group name and gene index
- `object = pgVirtual,name = missing,organism = missing,group = numeric,ind = numeric`: Remove gene based on gene group and gene index

Note

Required for subclasses of pgVirtual in order to extend the class system of FindMyFriends

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)
nGenes(testPG)

# Remove gene number 6
removeGene(testPG, ind=5)

# Remove all genes from organism 'AE017244'
removeGene(testPG, organism='AE017244')

# Remove first gene in gene group 10
removeGene(testPG, group=10, ind=1)
```

reportGroupChanges *Reports the change in grouping*

Description

This function inspects gene grouping before and after a change and reports on the changes. If newGrouping is missing it reports on the last performed comparison; optionally writing it to a file if 'file' is specified.

Usage

```
reportGroupChanges(newGrouping, oldGrouping, file)
```

Arguments

newGrouping	An integer vector as produced by seqToGeneGroup with the grouping after the change
oldGrouping	An integer vector as produced by seqToGeneGroup with the grouping before the change
file	A file to write

Value

This function is called for its side effects

Examples

```
# Show latest changes in grouping
reportGroupChanges()

# Alternatively write it to a file
reportGroupChanges(file = tempfile())
```

seqToGeneGroup *Get gene-to-genegroup relationship*

Description

This method returns the group membership for each gene in the pangenome as a vector of indices. Element 1 corresponds to gene 1 and the value is the index of the corresponding gene group. If gene groups have yet to be defined it returns a vector of length 0.

Usage

```
seqToGeneGroup(object)

## S4 method for signature 'pgInMem'
seqToGeneGroup(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer vector with an element for each gene in the pangenome.

Methods (by class)

- pgInMem: Gene to genegroup indexing for pgInMem subclasses

Note

Required for extending the class system of FindMyFriends

See Also

seqToOrg for gene-to-organism relationship

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Have a look at what the first six genes belongs to
head(seqToGeneGroup(testPG))
```

---

seqToOrg *Get gene-to-organism relationship*

Description

This method returns the organism membership for each gene in the pangenome as a vector of indices. Element 1 corresponds to gene 1 and the value is the index of the corresponding organism.

Usage

```
seqToOrg(object)

## S4 method for signature 'pgInMem'
seqToOrg(object)
```

Arguments

object	A pgVirtual subclass

Value

An integer vector with an element for each gene in the pangenome.

Methods (by class)

- pgInMem: Gene to organism indexing for pgInMem subclasses

Note

Required for extending the class system of FindMyFriends

See Also

seqToGeneGroup for gene-to-genegroup relationship

Examples

```
testPG <- .loadPgExample(withGroups=TRUE)

# Stored sequentially so the first will belong to organism 1
head(seqToOrg(testPG))
```

---

translated *Check the sequence type of the pangenome*

Description

This method checks whether the genes in the pangenome are on translated form (amino acid sequences) or not. A return value of FALSE only indicates that the storage mode for the genes is not an AAStringSet. While this leaves room for both RNA-, DNA- and BStringSet, only DNAStringSet makes much sense and is therefore assumed

Usage

```
translated(object)

## S4 method for signature 'pgVirtual'
translated(object)
```

Arguments

object	A pgVirtual subclass

Value

A boolean indicating whether genes are translated (TRUE) or not (FALSE)

Methods (by class)

- pgVirtual: Sequence type check for pgVirtual subclasses

Examples

```
testPG <- .loadPgExample()

# Genes are translated
translated(testPG)

# ... and therefore returned as AAStringSet instead of DNAStringSet
class(genes(testPG, subset=1))
```

`variableRegions` *Detect regions of high variability in the panchromosome*

Description

This method analyses the panchromosome and detects regions of local non-linearity. These regions often corresponds to areas with insertion/deletions, frameshifts or general high plasticity. It works by examining each vertice of the panchromosome with an out degree above 2 and detect cycles within the neighborhood of these vertices. Adjacent cycles are then joined together to form bigger groups of high variability.

Usage

```
variableRegions(object, ...)

## S4 method for signature 'pgVirtualLoc'
variableRegions(object, flankSize)
```

Arguments

`object`	A pgVirtualLoc subclass
`...`	parameters to pass on
`flankSize`	The size of the neighborhood around vertices with outdegree above 2 in where to search for cycles

Value

A list of variable regions. Each element contains the following elements:

type Either 'ins/den', 'frameshift', 'hub', 'plastic' or 'end'. ins/del are regions where the two outgoing vertices are directly connected. frameshift are regions where the two outgoing vertices are connected through two different routes, but not directly. hub are regions with more than two outgoing vertices. plastic are regions where the two outgoing vertices are connected through multiple different paths. end are regions with only one outgoing vertice.

members The gene groups being part of the region.

flank The outgoing vertices connecting the region to the rest of the panchromosome.

connectsTo The gene group(s) each flank connects to outside of the region

graph The subgraph of the panchromosome representing the region

Methods (by class)

- `pgVirtualLoc`: Variable region detection for all pgVirtualLoc subclasses

Examples

```
testPG <- .loadPgExample(geneLoc=TRUE, withNeighborhoodSplit=TRUE)

# Too heavy to include
## Not run:
regions <- variableRegions(testPG)
```

```
# Have a look at the first region
regions[[1]]

## End(Not run)
```

Index